بسم الله الرحمن الرحيم

مبادئ اساسية

١-١	المقدمة	Introduction
١-٢	هياكل البيانات	Data Structures
١-٣	انواع هياكل البيانات	Type of Data structures
١-٤	كيفية اختيار الهيكل البياني	Selection of Data Structures

كلية المعارف الجامعة

قسم علوم الحاسبات

بلال العلواني

bilal_z92@yahoo.com

١-١ المقدمة

من العوامل المهمة في معالجة البيانات والحصول على النتائج المطلوبة بطرق كفوءة هو ضرورة معرفة طرق تمثيلها واساليب التعامل مع هياكلها التمثيلية لذا فان هياكل البيانات لا تعني تمثيل البيانات في هياكل معينة بل قياس متطلباتها من حيث المساحة الخزنية (space)والوقت (time) اذ ان لكل طريقة مزايا تختلف عن غيرها مما يستوجب اختيار المناسب منها وفق التطبيق المعني.

تنقسم صيغة اتعامل مع الهياكل الى نوعين الاول هو الهيكل الفيزياوي ويقصد به المادي او الحيز الذي تخزن او تمثل فيه البيانات في ذاكرة الحاسوب(memory)التي نتعامل معها بصورة مصفوفة احادية من المواقع الخزنية.

اما الهيكل الثاني فهو الهيكل المنطقي وهو الشكل البرمجي او الاسلوب الذي يتعامل به المبرمج مع تلك البيانات. فمثلاً عند تعريف مصفوفة ثنائية A [4][5] فانها تمثل في ذاكرة الحاسوب في (٢٠) موقع متعاقب، والمبرمج عند استخدامه او عند تعامله مع بيانات هذه المصفوفة باعتبارها مكونة من اربعة صفوف وخمسة اعمدة كما نتعامل معها رياضياً ، فالوصول الى الموقع A[1][2] لا يعني البحث فيزياوياً في الصف الثاني والعمود الثالث لان مثل هذه الصور غير موجودة فيزياويا بل يجب البحث عن الموقع الثامن (بافتراض استخدام طريقة الصفوف لتمثيل المصفوفة في لغة C) ابتداءاً من اول موقع حدد لتمثيل المصفوفة اي ان المبرمج لم يكن معنيا بكيفية تمثيل بيانات المصفوفة في ذاكرة الحاسوب (التمثيل الفيزياوي) واستخدم خوارزمية الوصول الى العناصر البيانية للمصفوفة بصيغ برمجية معينة للتوصل الى الحل . ان وجهة نظر المبرمج هنا تمثل الهيكل المنطقي ، والترابط بين وجهة نظر المبرمج مع الهيكل الفيزياوي الفعلي فتعالجه لغة البرمجة.

١-٢ هياكل البيانات Data structures :

يمكن تعريف هياكل البيانات بانها: دراسة طرق الترابط بين نظرة المبرمجين للبيانات وعلاقة المعلومات بالاجهزة خصوصاً ذاكرة الحاسوب التي تخزن فيها البيانات.
هياكل البيانات تشمل طرق تنظيم المعلومات ، والخوارزميات الكفوءة في الوصول وطرق التعامل معها او تداولها (كالاضافة والحذف والتحديث والترتيب والبحث الخ) لذا فان الاهتمام لا ينحصر فقط باساليب الخزن وخوارزمياته لان الاهمية الحيوية هي قياس كلفة كل اسلوب من تلك الاساليب ومدى ملائمة استخدامها في الحالات المختلفة.

١-٣ انواع هياكل البيانات :

توفر لغات البرمجة الصيغ المناسبة لتعريف واستخدام العناصر البيانية ذات العلامة الواحدة (المنفردة) فمثلاً في لغة C تستخدم التعريفات :

```
int X;
float Y;
char S;
```

تمثل في ذاكرة الحاسوب ويتم التعامل معها بصيغ برمجية بسيطة مثل :

```
X=X+100
Y=Y+15.6
```

وتكاد تكون هذه الصيغ متوفرة في جميع لغات البرمجة بشكل قياسي شبه موحد . اما بالنسة للعناصر البيانية التي تتكون من عدة قيم بيانية فانها تحتاج لاستخدام هيكل بياني مختلف وفيما يلي ذكر لاهم تلك الهياكل البيانية.

١- المصفوفة ARRAY
٢- القيد RECORD
٣- الملف FILE
٤- الهياكل الخطية LINEAR STRUTURES
+الهياكل غير الموصولة NON-LINKED STRUCTURES
+المكدس STACK
+الطابورQUEUE
+الطابور الدائريCIRCULAR QUEUE
+الهياكل الموصولة LINKED STRUCTURES
+المكدس الموصول LINKED STACK
+الطابور الموصولLINKED QUEUE
٥- الهياكل غير الخطية NON-LINEAR STRUCTURES
+المخططات GRAPHS
+ المخطط المتجهDIRECTED GRAPH
+هيكل الشجرةTREE STRUCTURE
+ المخطط غير المتجهUNDIRECTED GRAPH

١-٤ كيفية اختيار الهيكل البياني المناسب:

لكل مجموعة من البيانات هنالك اكثر من طريقة لتنظيمها ووضعها في هيكل بياني معين ويتحدد ذلك وفق عدد من العوامل والاعتبارات لاختيار الهيكل البياني المناسب وهي:

١- حجم البيانات
٢- سرعة وطريقة استخدام البيانات
٣- الطبيعة الديناميكية للبيانات كتغييرها وتعديلها دورياً
٤- السعة الخزنية المطلوبة
٥- الزمن اللازم لاسترجاع اية معلومة من الهيكل البياني
٦- اسلوب البرمجة

المصفوفة ARRAY

٢-١ المصفوفة ARRAY

٢-٢ تمثيل المصفوفة الاحادية في الذاكرة Representation of one –dimensional Array

٢-٣ تمثيل المصفوفة الثنائية في الذاكرة Representation of two –dimensional array

٢-٣-١ طريقة الصفوف Row-wise

٢-٣-٢ طريقة الاعمدة Column- wise

٢-٤ تمثيل المصفوفات الثلاثية والرباعية الابعاد Representation of tree & four- dimensional array

٢-١ المصفوفة Array

هي عبارة عن مجموعة من المواقع الخزنية في الذاكرة تستخدم وتتصف بما يأتي:

١- جميع المواقع تكون من نوع بياني واحد ، حسب صيغة التعريف ،…int,char, float الخ .
٢- يمكن الوصول عشوائيًا (Randomiy accessed) إلى أي موقع من مواقعها دون الاعتماد على أي موقع في المصفوفة فمقدار الوقت المطلوب للوصول إلى أي موقع هو مقدار ثابت .
٣- مواقع عناصر المصفوفة تبقى ثابتة ولاتتغير أثناء التعامل مع أي من عناصر المصفوفة.
٤- تمثل المصفوفة في مواقع متعاقبة في الذاكرة .

٢-٢ تمثيل المصفوفة الأحادية في الذاكرة

في لغة ++C تعرف هذه المصفوفة كالآتي :

```
char x[N];
```

وهذا يعني تعريف هيكل بياني يستوعب مجموعة من العناصر البيانية عددها (N) مثلا باسم بياني واحد
هو (X) ويستخدم الدليل(index) للوصول الى العنصر البياني المطلوب، وتتراوح قيمة الدليل 0<=I<=N-1 وبموجب هذا التعريف يحدد مترجم اللغة (COMPILER) المنطقة الخزنية لاستيعاب مجموعة العناصر البيانية ويكون الموقع الاول مخصصا للعنصر الاول في المصفوفة وهو ما يطلق عليه عنوان البداية (BA)Base Address وليكن افتراضا هو (٥٠٠) اما العنصر الثاني للمصفوفة فيكون عنوانه بعد عنوان البداية مباشرة اي (٥٠١) وهكذا بقية العناصر بالتتابع،تجدر الاشارة هنا الى ضرورة ضرب الناتج المحسوب لقيمة موقع العنصر المطلوب بحجم (size) التعريف للعنصر البياني قبل جمعه مع عنوان البداية، ويستخدم الدليل I بقيمته التي تتراوح بين 0<=I<=N-1 ،نسبة الى موقع البداية (٥٠٠) باستخدام العلاقة التالية :

Location (X[I] = Base address + (I*Size)

فاذا كان المطلوب تحديد عنوان (موقع) العنصر الرابع في المصفوفة اي I= 3 فان

Location (X[3]) = 500+ (3*1)

= 500+3

=503

اي ان موقع (عنوان) العنصر الرابع هو الخلية (٥٠٣) لان العنصر الاول في الموقع ٥٠٠ والعنصر الثاني في الموقع ٥٠١ و العنصر الثالث في الموقع ٥٠٢ و العنصر الرابع في الموقع ٥٠٣فعندما يتضمن البرنامج اية اشارة او تعامل مع عناصر المصفوفة في اي ايعاز cin>>x[i],cout<<x[i] او غيرها فان المترجم يعتمد العلاقة المشار اليها اعلاه لتحديد الموقع المطلوب .

٢-٣ تمثيل المصفوفة الثنائية في الذاكرة

هنالك طريقتان لتمثيل المصفوفة الثنائية هما طريقة الصفوف (Row_wise method) وطريقة الاعمدة

(column _wise method) لناخذ التعريف التالي للمصفوفة :

int A[M][N];

وهذا يعني تعريف هيكل بياني اسمه A يستوعب مجموعة من العناصر البيانية عددها (M*N) ويستخدم دليلين للوصول الى العنصر البياني المطلوب وهما:

0<= I < M لتحديد الصف الذي فيه العنصر

0 <= J < N لتحديد العمود الذي فيه العنصر

فمثلا العنصر A[3][5] حيث I=3 , J=5 سيعني العنصر الذي يقع في السطر الرابع والعمود السادس ويعتمد مترجم اللغة (COMPILER) احدى الطريقتين الاتيتين لتمثيل هذه المصفوفة :-

٢-٣-١ طريقة الصفوف ROW – WISE METHOD

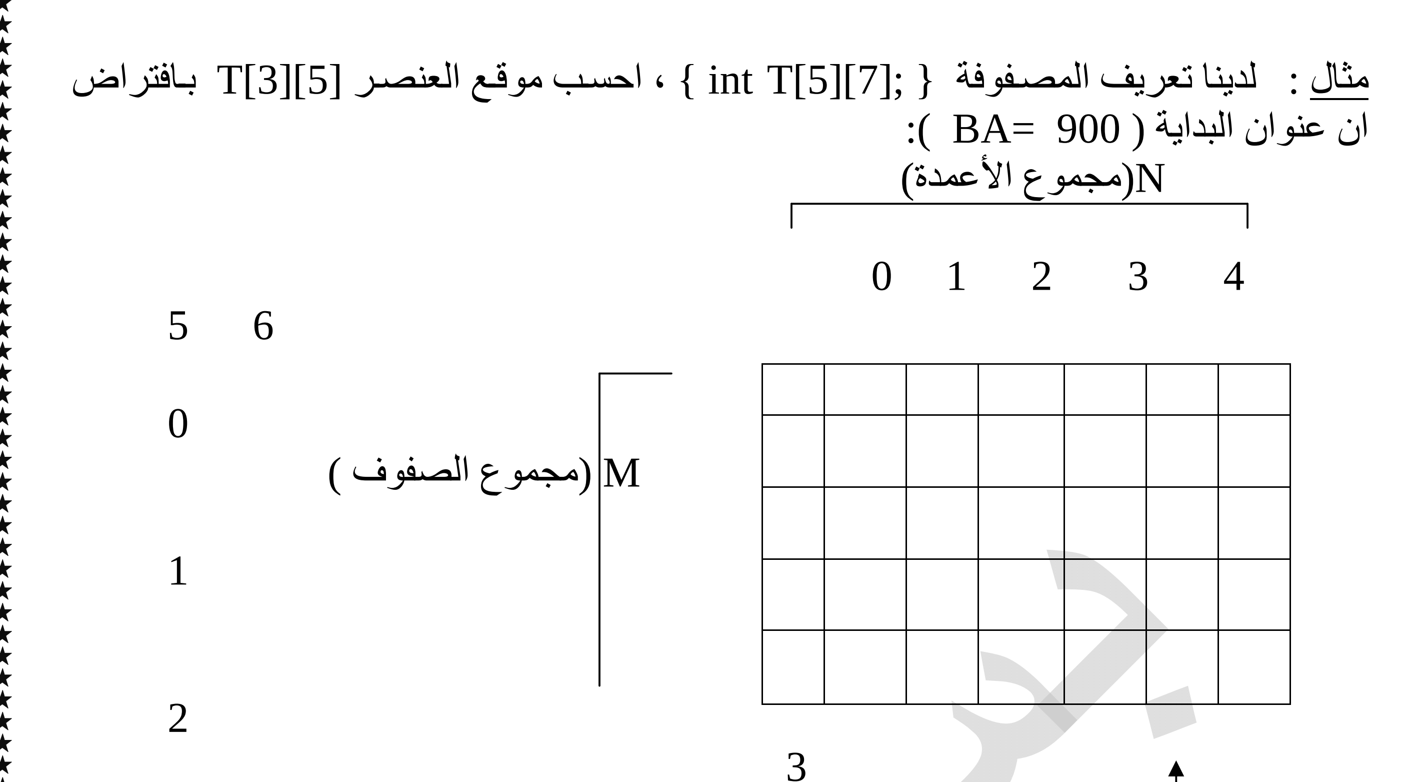
حيث تؤخذ جميع عناصر الصف الاول (I=0) للمصفوفة وتخزن في الذاكرة ابتداءا من موقع البداية (BASE ADDRESS) وليكن ٧٠٠، فالعنصر A[0][0] يخزن في الموقع BA اي ٧٠٠ ،والعنصر A[0][1] يخزن في الموقع BA+1*2 اي 702 ،والعنصر A[0][2] يخزن في الموقع BA+2*2 اي 704 ،ثم تؤخذ جميع عناصر الصف الثاني (I=1) للمصفوفة وتخزن في الذاكرة ابتداءا من الموقع الذي يلي اخر مواقع الصف الاول وتخزن جميع عناصر الصف الثالث (I=2) للمصفوفة في الذاكرة ابتداءا من الموقع الذي يلي موقع اخر عنصر من عناصر الصف الثاني وهكذا ..

ولهذا فان احتساب موقع العنصر A[I][J] يكون وفق العلاقة التالية :-

LOCATION (A[I,J]) = BASE ADDRESS + [(N* I + j)*size

حيث N تمثل مجموع اعمدة المصفوفة و I عدد الصفوف السابقة لموقع العنصر المطلوب و Jعدد الاعمدة السابقة لموقع العنصر المطلوب و size يمثل حجم العنصر،وهذه العلاقة هي التي يحتسب المترجم بموجبها موقع العنصر المطلوب معالجته بموجب كل ايعاز من ايعازات البرنامج .

مثال : لدينا تعريف المصفوفة { ;int T[5][7] } ، احسب موقع العنصر T[3][5] بافتراض ان عنوان البداية (BA= 900):



بما ان المطلوب هو العنصر T[3][5] فهذا يعني ان العنصر يقع في الصف الرابع(I=3)والعمود السادس (J=5) وبما ان مجموع صفوف المصفوفة (M=5) ومجموع اعمدة المصفوفة (N=7) لذا تصبح العلاقة عند التعويض فيها كما ياتي :

LOCATION (T[3][5]) =BA+[(7* 3 +5)*size

=900+[(7*3+5)*2]

=900 +[(21 +5)*2]

= 952

٢-٣-٢ <u>طريقة الاعمدة COLUMN – WISE METHOD</u>

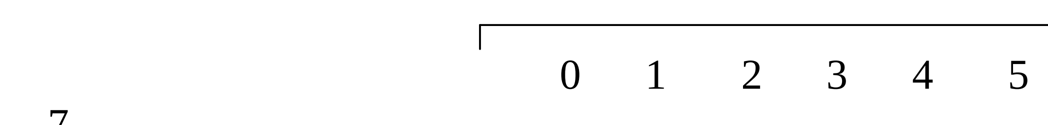
اذ تؤخذ جميع عناصر العمود الاول (J=0) للمصفوفة وتخزن في الذاكرة ابتداءا من موقع البداية (BASE ADDRES) وليكن ٢٠٠ ،فالعنصر A[0][0] يخزن في الموقع BA اي ٢٠٠، والعنصر A[1][0] يخزن في الموقع BA+٢ أي ٢٠٢ ،والعنصر A[2][0] يخزن في الموقع BA+ 4 أي ٢٠٤ ، ثم تؤخذ جميع عناصر العمود الثاني j= 1 للمصفوفة وتخزن في الذاكرة ابتداءاً من الموقع الذي يلي أخر مواقع العمود الأول ،وتخزن جميع عناصر العمود الثالث (j =2) للمصفوفة في الذاكرة ابتداءاً من الموقع الذي يلي موقع أخر عنصر من عناصر العمود الثاني وهكذا...

وعليه فأن احتساب موقع العنصر A[i][j] يكون وفق العلاقة التالية :--

Location (A[I][J]) =Base Address + [(M *J +I)*size]

مثال: اذا كان لدينا تعريف المصفوفة { int s[6][8]; } ، فما هو موقع العنصر s[4][6] عندما يكون عنوان البداية BA=300

N(مجموع الأعمدة)



M (مجموع الصفوف)

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								

S[4][6]

بما ان المطلوب هو العنصر s[4][6] فهذا يعني ان العنصر يقع في الصف الخامس(i=4) والعمود السابع (j=6) وبما ان مجموع صفوف المصفوفة(N=8) فالعلاقة تصبح:-

Location(s[4][6])=BA+(6*6+4)*2
=300+40*2
=300+80
=380

٢-٤ تمثيل المصفوفات الثلاثية والرباعية الابعاد:

بنفس الطريقة يمكن وضع الصيغة العامة لتحديد موقع العنصر للمصفوفة ذات الأبعاد الثلاثية أو الأربعة .

المصفوفة الثلاثية :int X[M][N][R] يكون احتساب موقع العنصر X [I][J][K] كالاتي:-

#طريقة الصفوف :

Location (X[I][J][K]) = BA+[(M*N*K+N*I+J)*size]

#طريقة الاعمدة:

Location (X[I][J][K) =BA+[(M*N*k+M*j+I)*size]

المصفوفة الرباعية :int Y[M][N][R][P] يكون احتساب موقع العنصر Y[I][J][K][L] كالاتي :

#طريقة الصفوف :

Location (Y[I][J][K][L])=BA+[(M*N*R*L+M*N*k+N*I+j)*size]

#طريقة الاعمدة:

Location (Y[I][J][K][L])=BA+[(M*N*R*L+M*N*K+M*J+I)*size]

تمرين : لديك المصفوفة الثلاثية التالية: { int TAB[8][5][7]; } ،احسب موقع العنصر TAB[4][2][5] في كل من طريقة الصفوف وطريقة الاعمدة اذا كان عنوان البداية base address=900

الحل:

The dimensions of TAB are : M=8,N=5,R=7

To compute the location of the element TAB[4][2][5]

This means the indices are : I=4,j=2, k=5

طريقة الصفوف Row-Wies

Location (TAB [I][J][K]) =BA+[(M*N*R*L+M*N*k+N*I+j)*size]

Location (TAB[5][3][6])=900+(8*5*5+5*4+2)*2

=900+(200+20+2)*2

=900+222*2

=1344

طريقة الأعمدة Column-Wise

Location(TAB[i][j][k])=BA+[(M*N*R*L+M*N*K+M*J+I)*size]

Location(TAB[5][3][6])=900+(8*5*5+8*2+4)*2

=900+(200+16+4)*2

=900+440

=1340

تمرين:لديك المصفوفة الرباعية التالية:- { intBOB[4][9][6][8]; } ،احسب موقع العنصر BOB[2][6][3][4] بطريقة الصفوف و طريقة الأعمدة إذا كان عنوان البداية Base Address=415

الحل:

The dimensions of BOB are: M=4,N=9,R=6,P=8

To compute the Location of the element BOB[2][6][3][4]

This means the indices are: i=2,j=6,k=3,l=4

طريقة الصفوف Row-Wise

Location(TAB[i][j][k][l])=BA+[(M*N*R*L+M*N*k+N*I+j)*size]

Location(TAB[3][7][4][5])=415+(4*9*6*4+4*9*4+9*3+7)*2

=415+(864+144+27+7)*2

=415+1042*2

=2499

Column-Wise طريقة الأعمدة

Location(TAB[i][j][k][l])=BA+[(M*N*R*L+M*N*K+M*J+I)*size]

Location(TAB[3][7][4][5])=415+(4*9*6*5+4*9*4+4*7+3)*2

=415+(1080+144+28+3)*2

=2925

اسئلة الفصل

1-What do we mean by data structures?Explain that in detail?
2-What are the classifications of data structures?
3-What the main factors for selection the required data structure?
4-What are the characteristics of the array?
5-Let int x[50];
What is the address of the element x[33] if the base address (BA=970)
6-Let int a[M][N];
How you can compute the address of the general element a[i][j] using
row-wise method?
7-Let int x[a][b][c];
what is the address of the general element x[i][j][k] using column-wise
method?
8-Let char s[7][10][8];

compute the location of the element s[5][9][3] using row-wise and
column-wise methods when the base address is 1200.

9-Let int a[9][5][8][6];
compute the location of the element a[8][3][6][4] using row-wise and
column-wise methods when the base address is 950.



الفصل الثالث

المكدس والطابور

1-3 القائمة الخطية: Linear list

1-1-3 انواع القوائم الخطية : Type of linear

2-3 المكدس: Stack

تمثيل المكدس باستخدام Array representation of stack
المصفوفة: 3-2-1

خوارزميات stack`s algorithms
3-2-2 المكدس:

البرامج لتنفيذ عمليات المكدس: Stack’s subprograms
3-2-3 الفرعية

3-2-4 تمثيل المكدس باستخدام القيد: **Record representation of stack**

أهم تطبيقات المكدس: Stack’s Application
3-2-5

3-3 الطابور: Queue

تمثيل الطابور باستخدام Array representation of queue
3-3-1 المصفوفة
خوارزميات : Queue `s algorithms
3-3-2 الطابور

3-3-3 البرامج الفرعية لتنفيذ عمليات الطابور: Queue’s Subprograms

3-3-4 تمثيل الطابور باستخدام القيد: Record Representation of Queue

3-3-5 تطبيقات الطابور: **Queue’s Applications**

3-4 الطابور الدائري: Circular Queue(CQ)

5-3 الطابور المزدوج: Double Ended Queue

1-3 القائمة الخطية linear list

هي مجموعة من العناصر البيانية (items ,nodes, elements) المتسلسلة و المرتبة تربط عناصرها علاقة تجاور بحيث يسبق كل عنصر عنصرا اخر عدا العنصر الأول الذي لا يسبقه عنصر و العنصر الأخير الذي لايليه عنصر فلو مثلنا كل عنصر على شكل عقدة (node) فان القائمة تصبح مجموعة من العقد(n).

x[1].x[2].x[3]……..x[k-1].x[k].x[k+1]………….x[n]

فالعقدة الاولى هي x[1] و العقدة الأخيرة هي x[n] اما العقدة x[k] عندما 1<=k<=n فان العقدة التي تسبقها هي x[k-1] والتي تليها هي x[k+1] .

ان كل مجموعة من البيانات و المعلومات يمكن تسميتها قائمة (list)فمثلا:

@ مجموعة أسماء طلبة كلية ما مرتبة حسب الحروف الابجدية.

@ مجموعة أسماء المشتركين في دليل الهاتف مرتبة وفق نسق معين.

1-1-3أنواع القوائم الخطية:

أ-القوائم غير الموصولة Non-Linked-List

وهي القوائم التي لا تستخدم المؤشرات و تكون على شكل بيانات متتابعة و متجاورة (sequential) و تستخدم المصفوفات في تمثيلها.كما يستخدم هذا النوع عند معالجة البيانات التي لا تتعرض للتغيير كثيرا لصعوبة عمليات الحذف و الإضافة إذ قد تكون المواقع التالية في ذاكرة الحاسوب مشغولة أصلا مما يتعذر استخدامها لأغراض الحذف و الإضافة.

ب-القوائم الموصولة Linked List

وهي القوائم التي تستخدم المؤشرات (pointers)لتسهيل عمليات الإضافة والحذف والتعديل إذ يكون لكل عنصر مؤشر يحدد موقع العنصر التالي ،ووجود المؤشرات يلغي الحاجة لخزن بيانات القائمة في مواقع خزنية متجاورة.

2-1-3 العمليات التي يمكن إجراؤها على القوائم الخطية:

يمكن تنفيذ عدد من العمليات (الفعاليات) على أي هيكل بياني عند معالجة بياناته و فيما يلي أهم أنواع هـــذه العمليات التي يمكن تنفيذ بعضها أو كلها حسب التطبيق.

1- البحث search : هي عملية بحث داخل الهيكل البياني بقصد الوصول إلى عنصر (عقدة) معين فيه بموجب قيمة أحد الحقول يسمى حقل المفتاح(key field)أي أن البحث يتم وفق المحتويات و ليس العنوان.

2-إدخال (إضافة)ِ Addition : لإضافة عنصر(عقدة)جديد إلى الهيكل البياني مثل تسجيل طالب جديد في المدرسة.

3-حذف Deletion: حذف عنصر(عقدة) من الهيكل البياني ،مثل نقل طالب إلى مدرسة اخرى.

4-دمج Merge: دمج بيانات هيكلين او اكثر لتكوين هيكل بياني واحد.

5-فصل Split: تجزئة بيانات هيكل بياني إلى هيكلين او أكثر.

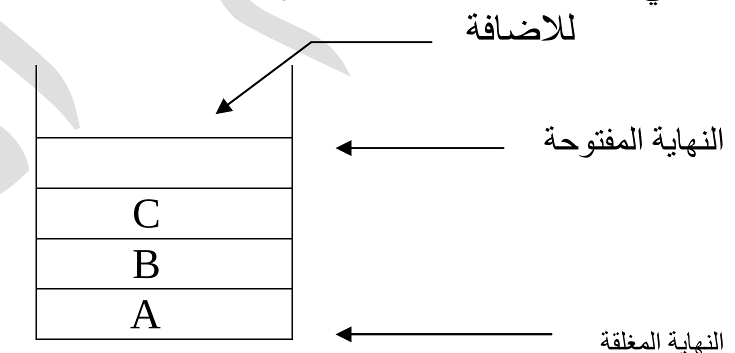
6-إحتساب Counting: احتساب عدد العناصر او العقد في الهيكل البياني.

7-نسخ Copying: نسخ بيانات الهيكل البياني الى هيكل بياني اخر.

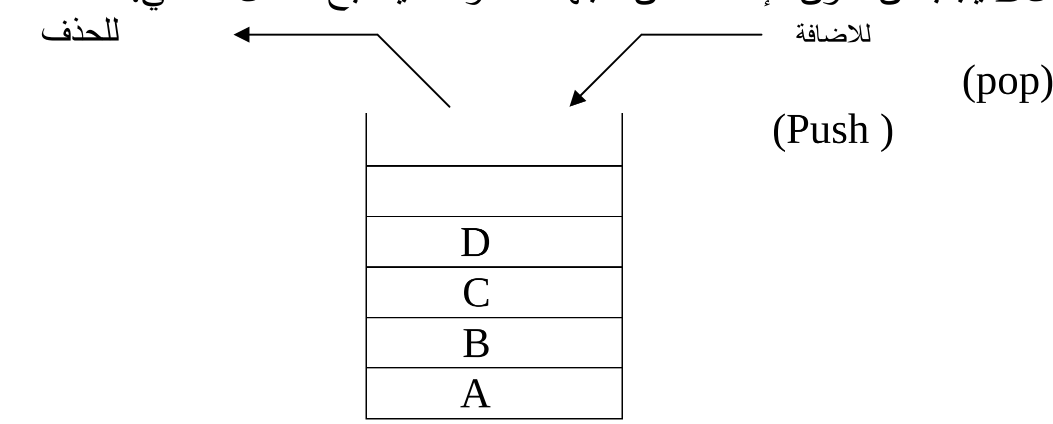
8-ترتيب Sort: ترتيب عناصر (عقد)الهيكل البياني وفق قيمة حقل (field)او مجموعة حقول.

9-الوصول Access: تتطلب أحيانا الحاجة للوصول إلى عنصر(عقدة)بياني في الهيكل البياني لعدة أغراض لاختباره مثلا أو تغييره...الخ.

2-3المكدس Stack :هو عبارة عن قائمة خطية تتم فيها عمليتي الإضافة والحذف من إحدى نهايتي القائمة وتكون النهاية الأخرى مغلقة.



لنأخذ المكدس الموضح في الشكل إذ نجده يحتوي على العناصر C,B,A و عند إضافة عنصر جديد مثلD يجب أن تكون الإضافة من الجهة المفتوحة ليصبح الشكل كالاتي:



وعند حذف عنصر من المكدس يجب ان نستخدم نفس الجهة المفتوحة فقط،اي نستطيع ان نأخذ العنصر (D) ثم نأخذ العنصر (C)بالتتابع ولانستطيع أن نأخذ العنصر (C)قبل أن نأخذ العنصر (D) مع ملاحظة ان العنصر (D) دخل أخيرا. ولهذا نستطيع أن نلخص عمل المكدس بالعبارة الآتية: (اخر من يدخل اول من يخرج) Last In First Out(LIFO)

كما انه لايمكن اخذ(حذف) عنصر من وسط عناصر المكدس إلا بعد حذف(إخراج) العناصر التي تسبقه من جهة النهاية المفتوحة مع التأكيد على أن النهاية الاخرى مغلقة ولا تستخدم أبدا. و تسمى عملية الإضافة الى المكدس (push)أو(Insertion)و عملية الحذف من المكدس (pop) او(Deletion).

<u>مثال</u>: نفرض (S) تعني (Stacking)اي ترمز لعملية إضافة عنصر الى المكدس و(U)تعني (Unstacking) اي ترمز لعملية حذف عنصر من المكدس وكانت مجموعة المدخلات للمكدس بالترتيب R,N,Y,B,M من اليمين، بين ما هي المخرجات بعد تنفيذ كل سلسلة من العمليات الآتية من اليسار الى اليمين:

أ-SSUUSUSUSU

ب-SSSUSUUSUU

<u>الحل</u>:

يقصد بترتيب المدخلات انه عند تنفيذ عملية إدخال عنصر الى المكدس فان اختيار العنصر يكون من تلك المدخلات بالتتابع اي نأخذ M أولا ثم B ،...وهكذا،ولا نستطيع اخذ العنصر N قبل العناصر السابقة له.

ا-

المدخلات	→ M	B			Y		N		R	
سلسلة العمليات	S →	S	U	U	S	U	S	U	S	U
المخرجات	→		B	M		Y		N		R

ب-

المدخلات	→ B	M	Y		N			R		
سلسلة العمليات	S →	S	S	U	S	U	U	S	U	U
المخرجات	→			Y		N	B		R	M

<u>مثال</u>: إذا كانت مجموعة مدخلات مكدس بترتيب 5,4,3,2,1 من اليمين إلى اليسار، بين أيا من المخرجات المبينة أدناه صحيحة وفق اسلوب عمل المكدس. (ترتيب المخرجات من اليسار الى اليمين)

أ- 1 3 5 4 2

ب-5 1 3 2 4

ج- 3 2 1 5 4

د- 2 1 5 3 4

<u>الحل</u>:

<u>الفرع</u> أ: المخرجات المطلوبة (2,4,5,3,1)

لاخراج العنصر (2) يجب أولا إدخال النصرين 2,1 اي أن تسلسل تنفيذ العمليات هو SSU أي ان محتويات المكدس تصبح :

1

ولاخراج العنصر(4) بعد العنصر(2) يجب إدخال العنصرين 4,3 اي ان تسلسل تنفيذ العمليات في هذه الحالة هو SSUSSU وتصبح محتويات المكدس :

3
1

ولإخراج العنصر (5) بعد العنصر(4) يجب إدخاله اولا ثم إخراجه أي ان تسلسل تنفيذ العمليات يكون SSUSSUSU وتصبح محتويات المكدس:

3
1

وفق حالة المكدس الحالية يمكن إخراج العنصرين 1 ثم 3 بالتتابع اي ان تسلسل تنفيذ العمليات هو SSUSSUSUUU .إذ يمكن الحصول على مثل هذه المخرجات إذا كان تسلسل العمليات بالصيغة الأخيرة مع الألتزام بترتيب المدخلات.

الفرع ب: المخرجات المطلوبة(4,2,3,1,5)

لاخراج العنصر(4) يجب اولا إدخال العناصر4,3,2,1 وفق سلسلة العمليات SSSSU وتصبح محتويات المكدس:

3
2
1

ولاخراج العنصر(2) من المكدس بحالته الحالية يجب إخراج العنصر(3) قبله لذا فان هذا التسلسل من المخرجات (4,2,3,1,5)لا يمكن تنفيذه.

الفرع ج: المخرجات المطلوبة(4,5,1,2,3)
يمكن إخراج العنصرين 5,4 بعد تنفيذ سلسلة العمليات الآتية:

المدخلات →	1	2	3	4		5	
سلسلة العمليات →	S	S	S	S	U	S	U
المخرجات →					4		5

و ستصبح محتويات المكدس:

3
2
1

و هنا سيتعذر إخراج العنصر (1) قبل العنصرين (2,3) لذا فان تسلسل المخرجات (4,5,1,2,3) غير صحيح.

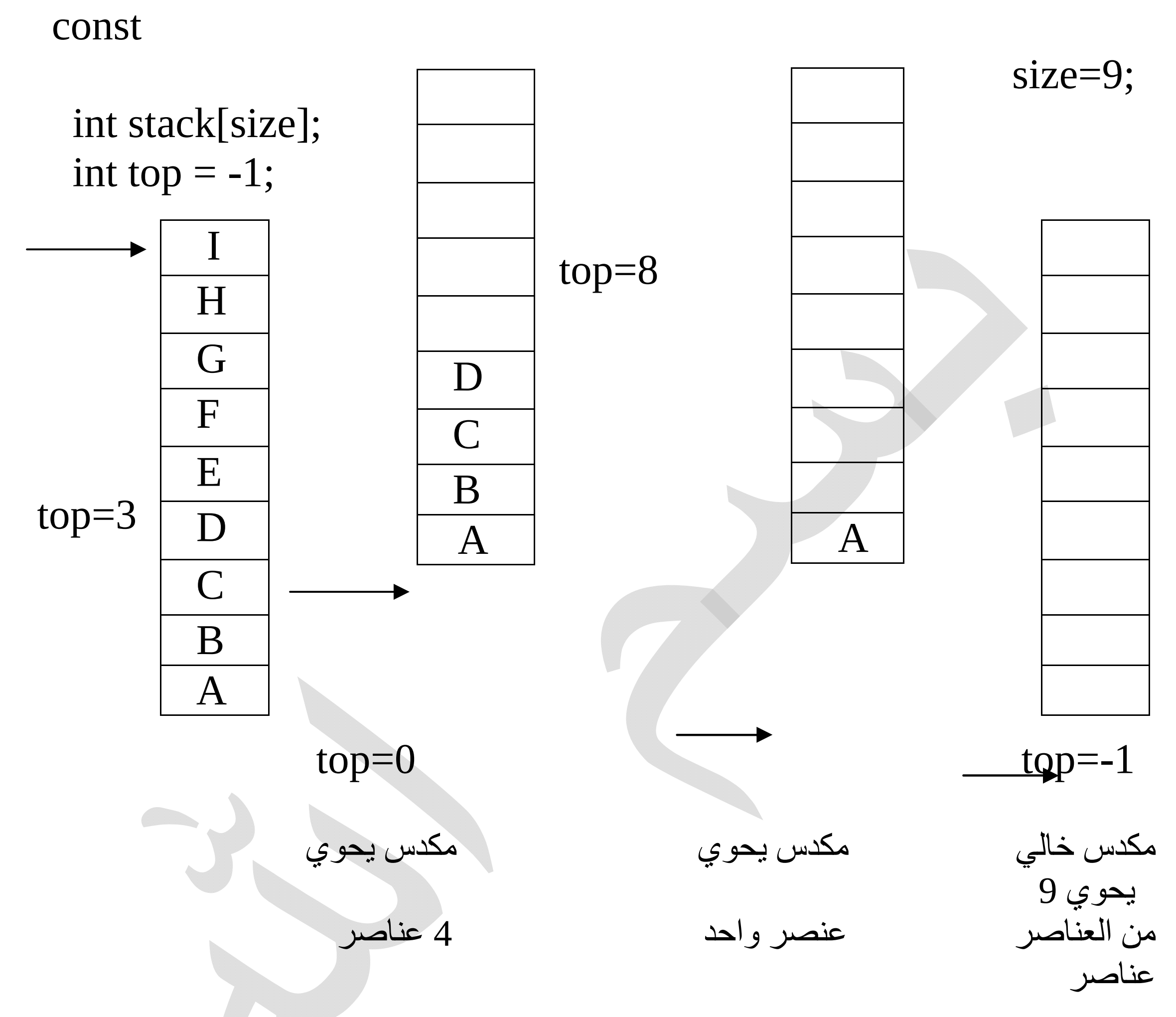
الفرع د: المخرجات المطلوبة(4,3,5,2,1)
يمكن الحصول على هذه المخرجات عند تنفيذ عمليات الإدخال و الإخراج بالتسلسل الآتي:

المدخلات →	1	2	3	4			5			
سلسلة العمليات →	S	S	S	S	U	U	S	U	U	U
المخرجات →					4	3		5	2	1

3-2-1 تمثيل المكدس باستخدام المصفوفة Array Representation Of Stack

يمكن تطبيق المكدس باستخدام مصفوفة احادية بالسعة المطلوبة (size) وبالنوع المناسب للبيانات (Data Type) التي ستخزن فيه (float , int ...الخ) مع استخدام متغير مستقل يدعى (Top) يستعمل كمؤشر يشير الى موقع اعلى عنصر في المكدس (موقع اقرب عنصر الى النهاية المفتوحة)وابتداء تكون قيمة المؤشر (Top=-1) عندما يكون المكدس خاليا من العناصر،و يعرف المكدس برمجيا كالآتي:

```
const int size=9;
int stack[size];
int top = -1;
```



عملية الإضافة للمكدس(push)

لتنفيذ عملية الإضافة بشكل صحيح نتبع الخطوات الآتية:

1-التحقق من كون المكدس غير مملوء (not full) أي أن المؤشرtop<size-1 لتجنب حالة الفيض (over flow) وتعذر تنفيذ عملية الإضافة.

2- تحديث قيمة المؤشر top=top+1 ليشير الى الموقع التالي (الفارغ).

3- إضافة العنصر الجديد في الموقع الجديد stack[top].

عملية الحذف من المكد (pop):

ان تنفيذ عملية حذف اي عنصر من المكدس يجب ان تكون وفق الخطوات الآتية:

1-التحقق بأن المكدس غير خال (not Empty) أي أن المؤشر top!=-1 لتجنب حالة الغيض (under flow) وتعذر تنفيذ عملية الحذف.

2- اخذ العنصر من الموقع الذي يشير اليه (top) وخزنه وقتيا في متغير مستقل

Item=stack[top];

3- تحديث قيمة المؤشر top=top-1 ليشير الى موقع العنصر التالي للعنصر الذي حذف.

ملاحظة:

يتضح أعلاه ان الخطوتين 3,2 في عملية الحذف معكوسة الترتيب عنها في عملية الإضافة.

2-2-3 خوارزميات المكدس Stack's Algorithms :

يمكن تصميم مجموعة من الخوارزميات لتغطية فعاليات المكدس و من ثم برمجتها و تمثيلها عمليا.

1-خوارزمية الإضافة push Algorithm

```
if          stack is full
Then        Over flow ← True
Else
            Over flow ← false
            Top ← Top+1
            Stack[top] ← New element
```

2- خوارزميات الحذف pop Algorithm

```
if          Stack is Empty

Then        Under flow ← True
Else
            under flow ← false
            element ← stack[top]
            Top=Top-1
```

3- خوارزمية ملء المكدس Stack full

هذه الخوارزمية للتحقق من هل المكدس مملوء أم لا اعتمادا على قيمة المؤشر (Top) قبل عمليات الإضافة

```
If          Top=size-1
Then        stackfull ← True
```

4- خوارزمية خلو المكدس Stack Empty

هذه الخوارزمية للتحقق من هل أن المكدس خال أم لا اعتمدا على قيمة المؤشر (Top) قبل عملية الحذف

```
If          Top=-1
```

Then stackempty ← True

5- خوارزمية إخلاء المكدس ClearStack

هذه الخوارزمية تستخدم لغرض تهيئة المكدس وإخلائه من العناصر بجعل قيمة المؤشر(top=-1)

Top ← -1

3-2-3 البرامج الفرعية لتنفيذ عمليات المكدس

ان تصميم برامج فرعية (functions, procedures) لكل فعالية او عملية من عمليات المكدس تساعد على تبسيط و توضيح كيفية برمجة تلك العمليات ومن ثم تجميعها في برنامج واحد تتوفر فيه صفات البرمجة المهيكلة ويكون واضحا للقراءة وسهل الفهم و المتابعة و التحديث والتطوير.

ونفترض وجود التعريف التالي في مقدمة البرنامج لتكون البرامج الفرعية اللاحقة صحيحة

```
#include<iostream.h>
#include<stdlib.h>
const size=20;
int stack[size];
int top;
int item;
```

1- برنامج فرعي لاخلاء المكدس

```
void clearstack()
{
 top=-1;
}
```

لاحظ عدم الحاجة للمرور على جميع مواقع المصفوفة وجعلها مساوية للصفر والاكتفاء بجعل المؤشر

(top=-1) وهذا البرنامج الفرعي يستدعى في بداية التعامل مع برامج المكدس لجعله خاليا.

2- برنامج فرعي للتحقق من إمتلاء المكدس

```
int fullstack()
{
 if(top>=size-1)
 return(1);
 else return(0);
}
```

هذه الدالة يكون المخرج لها بموجب قيمة المؤشر(top) هو اما 1 (true) عندما يكون المكدس مملوء وتكون قيمته 0 (false) عندما يكون المكدس غير مملوء. والبرنامج الفرعي (fullstack)يستدعى داخل البرنامج الفرعي(procedure push) لينفذ عملية الإضافة.

3- برنامج فرعي للتحقق من خلو المكدس

```
int emptystack()
```

```
{
 if(top==-1)
 return(1);
 else return(0);
}
```

هذه الدالـة يكـون المخـرج لهـا بموجـب قيمـة المؤشـر(top) هوامـا 1 (true) عنـدما يكـون المكــدس خاليــا و 0 (false) عنــدما يكــون المكــدس غيــر خــال وهــذا البرنــامج الفرعــي (emptystack) يستدعى داخل البرنامج الفرعي (procedure pop) الذي ينفذ عملية الحذف.

4-برنامج فرعي لاضافة عنصر واحد الى المكدس

```
void push(int item)
{
 if(fullstack())
 {
  cout<<"error...the stack is full"<<endl;
  cout<<"press any key to exit"<<endl;
   getch();
   exit(0);
   }
  else
  {
   top=top+1;
   stack[top]=item;
  }
 }
```

هذا البرنـامج الفرعـي يضـيف عنصـر واحـد (item) للمكـدس ويمكـن اسـتدعائه فـي البرنـامج الرئيسي (main program) بأي عدد من المرات بأستخدام احد ايعازات التكرار مثل (for…Do while) الذي يتضمن قراءة العنصر (item) ثم استدعاء البرنـامج الفرعـي (push) لاضـافته الـى المكدس.

5- برنامج فرعي لحذف عنصر واحد من المكدس

```
void pop()
{
 if(emptystack())
 {
   cout<<"error…the stack is empty"<<endl;
   cout<<"press any key to exit"<<endl;
   getch();
   exit(0);
```

```
    }
   else
   {
    item=stack[top];
    top=top-1;
   }
  }
```

هـذا البرنـامج الفرعـي يأخـذ العنصـر الـذي يشـير اليـه(top) وينسـخه فـي المتغيـر (item) لاستخدامه لاحقا بمعالجة معينة لتحقيق الغرض الذي من اجله سحب هذا العنصر من المكدس.
ولغرض حذف أو سحب اكثر من عنصـر من المكدس بصـورة متتابعة فان هذا البرنـامج يستدعى باي عدد من المرات وفي اي موقع من البرنـامج الرئيسي،باستخدام احدى صيغ التكرار المشار اليها.

4-2-3 تطبيق المكدس باستخدام القيد:

فـي التطبيـق السـابق باسـتخدام المصـفوفة ورد تعريـف المؤشـر(top) كمتغيـر مسـتقل عـن المصفوفة التي تمثل المكدس،إلا اننا هنا نستخدم القيد (Record) في تمثيلهما معا كهيكل بياني واحد حيث يتكون القيد من جزأين الأول يمثل المكدس وهوعلى شكل مصفوفة والجزء الثاني هو حقل يمثل المؤشر(top) ويعرف كالاتي:

```
int item;
const size=10;
struct stack
{
  int top;
  int element[size];
 }st;
st.top=-1;
```

stack:

A	B	C							3

element هذا الجزء هو   top

st.element[index] ويستخدم باسم   st.top ويستخدم باسم

لاضافة عنصر جديد لهذا المكدس نتبع الخطوات التالية:

1-نحدث قيمة المؤشر (top) الذي هو حقل في القيد stack ليصبح (3) :

st.top=st.top+1;

2-نضيف العنصر الجديد(D)في الموقع الجديد(3) :

st.element[st.top]=D;

وبهذا يصبح المكدس بالصورة التالية:

stack:

A	B	C	D						3

st. elements   st.top

st.element[3]

تمرين:اعد كتابة البرنامج الفرعي (pop) لحذف عنصر من المكدس باستخدام القيد .

```
void pop()
{
  if(emptystack())
  {
   cout<<"error…the stack is empty"<<endl;
   cout<<"press any key to exit"<<endl;
   getch();
   exit(0);
  }
   else
  {
    item=st.element[st.top];
    st.top--;
    }
}
```

تمرين:اكتب برنامجا فرعيا لاضافة ثلاثة عناصر من الأعداد الصحيحة الى المكدس (set) الذي سعته(20)

الحل: ان المكدس المطلوب يمثل بالمصفوفة(set) وسعتها(20) ونوع البيانات (int)

```
int set[20];
int top;
void push3()
{
  int i;
  for(i=0;i<3;i++)
   {
    top++;
    if(top==20)
     {
      cout<<"error...the stack is full"<<endl;
       cout<<"press any key to exit"<<endl;
       getch();
        exit(0);
```

```
        }
        else
        {
          cout<<"enter the element"<<endl;
          cin>>set[top];
         }
      }
    }
```

تضمن هذا البرنامج الفرعي (procedure) خطوة التحقق من امتلاء المكدس داخل ايعاز التكرار لينفذ عند كل عملية اضافة مع ان سعة المصفوفة(20) والسبب اننا لا نعرف عدد عناصر المكدس قبل الاضافة.

مثال:المكدس(table) بسعة(30) عنصر يحتوي على اربعة عناصر D,C,B,A ، اكتب برنامجا فرعيا (procedure) لاضافة (8)عناصر اخرى.

الحل:ان المكدس المطلوب يمثل بالمصفوفة(table) وسعتها (30) ونوع بياناته هو (char)

```
char table[30];
int top;
void push8()
{
  int i;
  top=3;
  for(i=0;i<8;i++)
    {
      top++;
      cout<<"enter new element\n";
      cin>>table[top];
    }
 }
```

في هذا البرنامج الفرعي (procedure) لم نضع خطوة التحقق من امتلاء المكدس لكونها غير ضرورية لان سعة المكدس هي (30) ويحتوي على اربعة عناصر فقط والاضافة المطلوبة هي(8) فقط لذا فان المكدس لن يصل الى حالة الأمتلاء.

مثال:اكتب برنامج فرعي لحذف (4)اعداد حقيقية من المكدس(BOB)الذي سعته (15)عنصر.

الحل:ان المكدس المطلوب يمثل المصفوفة(BOB) بسعة(15) عنصر ونوع البيانات (float)

```
float BOB[15];
int top;
float item;
```

```
void pop4()
{
  int i;
  for(i=0;i<4;i++)
    {
     if(top==-1)
      {
        cout<<"error…the stack is empty"<<endl;
        cout<<"press any key to exit<<endl;
        getch();
        exit(0);
       }
        else
          {
            item=BOB[top];
            top--;
          }
     }
 }
```

5-2-3 أهم تطبيقات المكدس

1- معالجة البرامج التي تحتوي على برامج فرعية:

يستخدم المكدس بأهمية كبيرة من قبل المترجمات في معالجة البرامج التي تحتوي على برامج فرعية(functions & procedures) وتنظيم طريقة استدعائها وذلك بخزن عناوين الرجوع (Return Addresses) فعند استدعاء برنامج فرعي داخل البرنامج الرئيسي فان ذلك يتطلب خزن عنوان الايعاز التالي بعد ايعاز الاستدعاء لكي يستطيع البرنامج الرئيسي تنفيذ البرنامج الفرعي والعودة بشكل صحيح الى موقع الخطوة او الايعاز التالي لان عنوان هذا الموقع (Return-address) يكون مخزونا في المكدس.

لنفترض ان البرنامج التالي الذي يتضمن استدعاء عدد من البرامج الفرعية هي C,B,A

```
Begin {this is the main program}
100 CALL   A
102          ----
-----        ----
-----        200 CALL B        ----
-----        202              ----
-----        ----             300 CALL C
-----        ----             302
-----        ----             ----
```

```
-----          ----           ----
-----          ----           ----
-----          ----           ----
END
```

نلاحظ في هذا المثال:

أ-أن البرنامج الرئيسي يستدعي البرنامج الفرعي (A) الذي بدوره وفي داخله يستدعي البرنامج الفرعي(B) وبداخله استدعاء البرنامج الفرعي(C).

ب-لغرض التوضيح نفترض أن عناوين الايعازات كما يأتي:

ان عنوان ايعاز استدعاء (A) هو(100) أما عنوان الايعاز التالي له(Ret.Add) فهو (102) وعنوان ايعاز استدعاء(B) هو(200) اما عنوان الايعاز التالي له (Ret.Add) فهو (202) وعنوان ايعاز استدعاء(C) هو(300) أما عنوان الايعاز التالي له(Ret.Add) فهو(302).

ج-أن مترجم اللغة يستخدم المكدس في معالجة مثل هذا النوع من البرامج وبالطريقة التالية:

1-عند الوصول الى استدعاء البرنامج الفرعي (A) وقبل تنفيذ الأستدعاء يخزن عنوان الرجوع (102) في المكدس بعملية(push).

2-عند تنفيذ ايعازات البرنامج الفرعي (A) نجده يتضمن ايعاز استدعاء البرنامج الفرعي(B) وهذا يتطلب قبل تنفيذ الاستدعاء خزن عنوان الرجوع (202) في المكدس بعملية(push) اخرى.

3-عند تنفيذ ايعازات البرنامج الفرعي (B) نجده يتضمن ايعاز ايعاز استدعاء البرنامج الفرعي(C) وهذا يتطلب قبل تنفيذ الاستدعاء خزن عنوان الرجوع (302) في المكدس بعملية(push) اخرى.

4-عند انتهاء تنفيذ البرنامج الفرعي(C) فأن البرنامج الرئيسي يحتاج معرفة عنوان الرجوع الذي سبق خزنه في المكدس ويتم ذلك من خلال تنفيذ عملية(pop) لاخراجه وتنفيذ الإيعاز الموجود في ذلك العنوان وما بعده داخل البرنامج الفرعي(B).

5-عند انتهاء تنفيذ ايعازالبرنامج الفرعي (B) فأن البرنامج الرئيسي يحتاج معرفة عنوان الرجوع الذي سبق خزنه في المكدس ويتم ذلك من خلال عملية(pop) لاخراجه وتنفيذ الإيعاز الذي في ذلك العنوان وما بعده داخل البرنامج الفرعي (A).

6- عند انتهاء تنفيذ ايعازالبرنامج الفرعي (A) فأن البرنامج الرئيسي يحتاج معرفة عنوان الرجوع الذي سبق خزنه في المكدس ويتم ذلك من خلال عملية(pop) لاخراجه وتنفيذ الإيعاز الذي في ذلك العنوان وما بعده داخل البرنامج الرئيسي.

7-يستمر البرنامج في تنفيذ الايعازات التالية بصورة اعتيادية بعد ان انتهت البرامج الفرعية ولم يعد المكدس يحوي شيئا(اي خاليا).

تمرين:وضح بالرسم جميع حالات المكدس عند تنفيذ البرنامج التالي:

```
Begin{main program}
100  CALL     X
102  --------
200  CALL     Y
202  --------
-----          400  CALL P
-----          402  --------
```

```
-----          -----     600   CALL  R
-----          -----     602    ------
-----          -----     -----   700   CALL  S
-----          -----     -----   702    -------
-----          500  CALL  Q
-----          502   ------
300            CALL  Z
302            -------
End.
```

2-استخدام المكدس في معالجة التعابير الحسابية

من المعروف ان التعابير الحسابية تكتب بثلاث صيغ هي:

1-صيغة Infix notation : حيث ان اشارة العملية الحسابية تتوسط العوامل مثل:-X/20,A-B,3+4 و هذه هي الصيغة الاعتيادية.

2-صيغة Prefix Notation: إذ تسبق إشارة العملية الحسابية العوامل مثل: / X 20,- A B,+ 3 4 وتسمى(Polish Notation)

3-صيغة Postfix Notation :إذ تلحق إشارة العملية الحسابية العوامل مثل: X 20 /,A B -,3 4 + وتسمى Reverser Polish Notation (RPN) لانها عكس الحالة الثانية (Polish Notation).

ملاحظة: لتنفيذ أي تعبير حسابي مكتوب بصيغة(Infix) فان العمليات تنفذ من اليسار الى اليمين و حسب أعلى أسبقية للعملية الحسابية وهي:

الأسبقية	نوع العملية الحسابية
4	^(power),Unary(-),Unary(+),Not
3	* , / , AND, DIV , MOD
2	+ , - , OR
1	= , < , > , != , <= , >=

وتستخدم الأقواس عند الحاجة إلى تغيير أسبقيات التنفيذ وتسلسل الخطوات،وبذلك تنفذ اولا ويعامل ما بداخلها على انه تعبير حسابي مستقل.

ان البرامج التي تتضمن تعابير حسابية بصيغة(Infix) يقوم المترجم (compiler) بتحويلها الى صيغة(postfix) باستخدام المكدس وفق الخوارزمية الاتية:

خوارزمية تحويل صيغة (Infix) إلى (Postfix) بأستخدام مكدسين

1- نستخدم مكدسين،المكدس الأول(ST1) لخزن المتغيرات (العوامل operands) وفي الخطوة الأخيرة ستتجمع فيه الصيغة النهائية(صيغة Postfix) و المكدس الثاني(ST2) يستخدم لخزن اشارات العمليات الحسابية(Operators).
2- نفحص التعبير الحسابي رمزا رمزا من اليسار الى اليمين.
3- عند كل رمز نقوم بما يأتي:

إذا كان الرمز	ينفذ ما يأتي:
+ احد العوامل(operand)	+ يخزن(push) في المكدس(ST1)
+ قوس ايسر	+ يخزن (push) في المكدس(ST2)
+ قوس ايمن	+ إخراج (pop) جميع الرموز من المكدس (ST2) بالتتابع ووضعها في المكدس (ST1) لغاية الوصول الى القوس الأيسرالذي يجب إخراجه وإهماله مع لقوس الأيمن.
+عملية حسابية(operator)	+ إخراج (pop) جميع العمليات الحسابية(ان وجدت) في المكدس(ST2)التي أسبقيتها أعلى أو تساوي أسبقية العملية الحسابية الحالية وخزنها في المكدس(ST1) (التوقف عن ذلك عند عدم تحقق الشرط)ومن ثم خزن العملية الجديدة في المكدس(ST2).

4- عند انتهاء كل رموز التعبير الحسابي يتم إخراج (pop) جميع الرموز المتبقية في المكدس(ST2) بالتتابع و خزنها(push) في المكدس (ST1) الذي يحوي الصيغة النهائية(Postfix).

مثال: حول التعبير الحسابي التالي من صيغة (Infix) الى صيغة(Postfix) باستخدام مكدسين.

a-b*(c+d)/(e-f)^g*h

رقم الخطوة	الرمز المدخل	المكدس الأول ST1	المكدس الثاني ST2
1	a	a	....
2	-	a	-
3	b	ab	-
4	*	ab	- *
5	(	ab	- * (
6	c	abc	- * (
7	+	abc	-* (+
8	d	abcd	- * (+
9	)	abcd+	- *

نلاحظ هنا عند ورود القوس الأيمن يتم إخراج (نقل) جميع العمليات الحسابية لغاية القوس الأيسر
من المكدس(ST2) إلى(ST1) مع إخراج القوس الأيسر ليهمل هو والقوس الأيمن.

- /	abcd+*	/	10
- / (	abcd+*	(	11
- / (	abcd+*e	e	12
- / (-	abcd+*e	-	13
- / (-	abcd+*ef	f	14
- /	abcd+*ef-	)	15
- / ^	abcd+*ef-	^	16
- / ^	abcd+*ef-g	g	17
- *	abcd+*ef-g^/	*	18

لان أسبقية الضرب (*) =< اسبقية الرفع (^) والقسمة(/)

- *	abcd+*ef-g^/h	h	19

هنا انتهت جميع المدخلات لذا ينقل المتبقي في المكدس(ST2)الى المكدس(ST1) بالتتابع ليصبح

......	abcd+*ef-g^/h*-	20

خوارزمية تحويل صيغة (Infix) الى (Postfix) باستخدام مكدس واحد

1- نستخدم مكدس واحد(ST) لخزن إشارات العمليات الحسابية(operators).
2- نفحص (نقرأ) التعبير الحسابي رمزا رمزا من اليسار الى اليمين.
3- عند كل رمز نقوم بما يأتي:-

إذا كان الرمز:	ينفذ ما يأتي:
+ احد العوامل(operand)	+ ينقل الى جملة المخرجات output string
+ قوس أيسر	+يخزن (push) في المكدس (ST).
+عملية حسابية(operator)	+إخراج (pop) جميع العمليات الحسابية(ان وجدت) في المكدس (ST) التي أسبقيتها أعلى اوتساوي أسبقية العملية الحسابية الجديدة وإضافتها الى جملة المخرجات (التوقف عن ذلك عند عدم تحقق الشرط). بعد ذلك تخزن(push) اشارة العملية الحسابية الجديدة في المكدس(ST).
+قوس أيمن	+ إخراج (pop) جميع إشارات العمليات الحسابية من المكدس وإضافتها بالتتابع الى جملة المخرجات لغاية الوصول الى القوس الأيسر في المكدس الذي يجب إخراجه وإهماله مع القوس الأيمن المقابل له.

4- عند انتهاء فحص (المرور على) جميع رموز التعبير الحسابي يتم إخراج (pop) جميع الرموز المتبقية في المكدس (ST) بالتتابع و إضافتها الى جملة المخرجات ليصبح الشكل النهائي لجملة المخرجات هو صيغة ال(Postfix) المطلوبة.

<u>تمرين</u>: حول العبارة الحسابية التالية من صيغة(infix) الى صيغة (postfix) باستخدام مكدس واحد .

y*m+(a^3/b-n)-d

	المكدس ST	الرمز المدخل	رقم الخطوة	output string
	...	y	1	y
	*	*	2	y
	*	m	3	ym
	+	+	4	ym*
	+(	(	5	ym*
	+(	a	6	ym*a
	+(^	^	7	ym*a
	+(^	3	8	ym*a3
^	+(/	/	9	ym*a3
	+(/	b	10	ym*a3^b
	+(-	-	11	ym*a3^b/
	+(-	n	12	ym*a3^b/n
	+	)	13	ym*a3^b/n-
	-	-	14	ym*a3^b/n-+
ym*a3^b/n-	-	d	15	+d
ym*a3^b/n-	....	...	16	+d-

احتساب قيمة (تنفيذ) التعبير الحسابي المحول الى صيغة Postfix

بعد أن يحول المترجم العبارة الحسابية من صيغة (infix) الى صيغة (Postfix) فأن احتساب قيمتها في المرحلة التالية يكون بموجب الخوارزمية المبينة أدناه بأستخدام مكدس واحد.

الخوارزمية

1- يستخدم مكدس واحد و ليكن (ST)

2- نفحص (نأخذ) التعبير الحسابي رمزا رمزا من اليسار الى اليمين ويعامل كالآتي:

إذا كان الرمز المدخل هو:	ينفذ ما يأتي:
+ أحد العوامل (operand)	+ يخزن (push) في المكدس(ST)
+ عملية حسابية (operator)	+ تنفذ هذه العملية على العاملين في أعلى المكدس(أي يتم إخراج العاملين من المكدس(ST)وتنفذ العملية عليهما) وتخزن النتيجة المتحققة في (ST)

3- عند انتهاء مدخلات التعبيرالحسابي فأن القيمة المتبقية في المكدس هي النتيجة النهائية للعبارة الحسابية.

مثال: لنأخذ العبارة الحسابية المكتوبة بصيغة(infix)   7+8- 6*3/2

عند تحويلها الى صيغة (postfix) تصبح -/2*78+63

ولاحتساب قيمة هذه العبارة بصيغتها الأخيرة نطبق خطوات الخوارزمية كالآتي:

رقم الخطوة	المدخلات	محتويات المكدس ST
1	7	7
2	8	7 8
3	+	15

لاحظ هنا تنفيذ عملية الجمع(+)على العاملين الموجودين في المكدس (7، 8) وخزن النتيجة (15) بدلهما في المكدس

رقم الخطوة	المدخلات	محتويات المكدس ST
4	6	15 6
5	3	15 6 3
6	*	15 18

لاحظ هنا تنفيذ عملية الضرب(*) على العاملين (6) ، (3) وخزن النتيجة (18) بدلهما في المكدس

رقم الخطوة	المدخلات	محتويات المكدس ST
7	2	15 18 2
8	/	15 9
9	-	6

لاحظ هنا تنفيذ عملية الطرح(-) على العاملين (15) ، (9) وخزن النتيجة (6) بدلهما في المكدس، ان القيمة المتبقية في المكدس (6) تمثل النتيجة النهائية لعملية احتساب قيمة العبارة الحسابية. تجدر الاشارة هنا الى ان هذه الخوارزمية يمكن استخدامها للتحقق من مدى صحة التعبير الحسابي المحول الى صيغة الـ Postfix وذلك في حالة بقاء عملية حسابية في التعبير مع عدم وجود معاملين في المكدس او العكس وهو عند انتهاء جميع الرموز مع وجود اكثر من قيمة في المكدس.

خوارزمية احتساب قيمة(تنفيذ) العبارة الحسابية بصيغة Infix :

من التطبيقات الأخرى للمكدس استخدامه في المفسرات(Interpreters) لاحتساب قيمة العبارة الحسابية المكتوبة بصيغة(Infix) بدون تحويلها الى صيغة(Postfix).

خطوات الخوارزمية

1- يستخدم مكدسان هما (ST1) لخزن العوامل الحسابية(operands) و(ST2) لخزن اشارات العمليات الحسابية(operaters).

2- تؤخذ رموز العبارة الحسابية بالتتابع واحدا بعد الآخر من اليسار الى اليمين.

3- حسب نوع الرمز نقوم بما يلي:

إذا كان الرمز:	ينفذ ما يأتي:
+احد العوامل(operand)	+ يخزن (push) في المكدس(ST1)
+عملية حسابية(operator)	+ اخراج(pop) بالتتابع جميع العمليات الحسابية(ان وجدت) في المكدس (ST2) التي أسبقيتها >= أسبقية العملية الحسابية الجديدة وتنفيذ كل منها على العاملين في قمة المكدس (ST1) وخزن النتيجة بدلهما في(ST1).

4- بعد انتهاء جميع رموز العبارة الحسابية نبدأ بتنفيذ جميع العمليات الحسابية المتبقية في المكدس ST2 بالتتابع

على كل عاملين في قمة المكدس (ST1) واحلال نتيجة تلك العملية محلهما في نفس المكدس(ST1) ونستمر بتكرارهذه الخطوة لحين خلو المكدس(ST2) وتكون آخر قيمة موجودة في المكدس (ST1) هي النتيجة النهائية.

ملاحظة هامة: في حالة وجود الاقواس في التعبير الحسابي يعامل ماموجود داخل الاقواس على انه تعبير حسابي مستقل، أي يتم دفع القوس المفتوح الى المكدس الثاني (ST2) واجراء فقرات الخطوة(٣) اعلاه لحين ورود القوس المغلق فيهمل ويستمر تطبيق الخوارزمية الى النهاية.

مثال : اوجد قيمة العبارة الحسابية الاتية المكتوبة بصيغة (INFIX) باستخدام مكدسين: 3+7*2-6

الخطوة	الرمز	ST1	ST2
١	٣	٣	......
٢	+	٣	+
٣	٧	3 7	+
٤	*	٣ ٧	+*
٥	٢	٣ ٧ ٢	+*
٦	-	١٧	-

لاحظ هنا تنفيذ عملية الضرب(*) على العاملين ٧، ٢ والنتيجة هي ١٤ لان اسبقيتها>= من العملية الجديدة الطرح – ثم الاستمرار في تنفيذ عملية الجمع+على النتيجة المتحققة ١٤ والقيمة ٣ لنحصل على ١٧ ولانتهاء العمليات الحسابية التي اسبقيتها > = اسبقية العملية الجديدة نخزن اشارة هذه العملية في المكدس ST2 .

٧	٦	٦	١٧

عند انتهاء جميع رموز العملية الحسابية المدخلة نبدا بتنفيذ العمليات الحسابية المتقبية في المكدس ST2 بالتتابع على محتويات المكدس ST1 وتصبح الخطوة الاخيرة.

٨ ...... ١١ ......

فالقيمة ١١ المتبقية في المكدس ST1 هي النتيجة النهائية

<u>مثال</u> : حول التعبير الحسابي التالي من صيغة INFIX الى صيغة POSTFIX باستخدام مكدسين

M:=X/6+(a-2*(b/3)^5+f)^2

st2 for operators	st1 for operands	input char	step no.
…..	M	M	1
:=	M	:=	2
:=	MX	X	3
:=/	MX	/	4
:=/	MX6	6	5
:=+	MX6/	+	6
:=+(	MX6/	(	7
:=+(	MX6/a	A	8
:=+(-	MX6/a	-	9
:=+(-	MX6/a2	2	10
:=+(-*	MX6/a2	*	11
:=+(-*(	MX6/a2	(	12
:=+(-*(	MX6/a2b	B	13
:=+(-*(/	MX6/a2b	/	14
:=+(-*(/	MX6/a2b3	3	15
:=+(-*	MX6/a2b3/	)	16
:=+(-*^	MX6/a2b3/	^	17
:=+(-*^	MX6/a2b3/	5	18
:=+(+	MX6/a2b3/5^*-	+	19
:=+(+	MX6/a2b3/5^*-F	F	20
:=+	MX6/a2b3/5^*-F+	)	21
:=+^	MX6/a2b3/5^*-F+	^	22
:=+^	MX6/a2b3/5^*-F+2	2	23
……	MX6/a2b3/5^*-F+2^+:=	…..	24

<u>مثال</u> : حول التعبير الحسابي التالي من صيغة INFIX الى صيغة POSTFIX باستخدام مكدسين (A>B)AND ((E-C>A)OR(G<F))

st2 for operators	st1 for operands	input char	step no.
(	…..	(	1
(	A	A	2
(>	A	>	3

(>	AB	B	4
…..	AB>	)	5
AND	AB>	AND	6
AND(	AB>	(	7
AND((	AB>	(	8
AND((	AB>	E	9
AND((-	AB>E	-	10
AND((-	AB>EC	C	11
AND((>	AB>EC-	>	12
AND((>	AB>EC-A	A	13
AND(	AB>EC-A>	)	14
AND(OR	AB>EC-A>	OR	15
AND(OR(	AB>EC-A>	(	16
AND(OR(	AB>EC-A>G	G	17
AND(OR(<	AB>EC-A>G	<	18
AND(OR(<	AB>EC-A>GF	F	19
AND(OR	AB>EC-A>GF<	)	20
AND	AB>EC-A>GF<OR	)	21
….	AB>EC-A>GF<OR AND	….	22

مثال : حول التعبير الحسابي التـالي من صيغة INFIX الـى صيغة POSTFIX باستخدام مكدسين:

A NOT (B OR Z OR NOT(G > E))

st2 for operators	st1 for operands	input char	step no.
…..	A	A	1
NOT	A	NOT	2
NOT(	A	(	3
NOT(	AB	B	4
NOT(OR	AB	OR	5
NOT(OR	ABZ	Z	6
NOT(OR	ABZ OR	OR	7

NOT(OR	A B Z OR	NOT	8
NOT (OR NOT(	A B Z OR	(	9
NOT (OR	A B Z OR G	G	10
			NOT(
NOT(OR	A B Z OR G	<	11
			NOT(<
NOT(OR NOT(<	A B Z OR G	E	12
NOT(OR	A B Z OR GE<	)	13
			NOT
NOT	A B Z OR GE<NOT OR	)	14
…..	A B Z OR GE < NOT OR NOT	….	15

٣- تطبيقات اخرى

يستخدم المكدس كهيكل لخزن المعلومات التي نحتاج استرجاعها بصورة معكوسة (بترتيب معكوس) والحالات التي تتطلب العودة الى موقع الخطوة السابقة BACK TRACKING وكمثال على ذلك مسائل المتاهة A
MAZING PROBLEMS فعند المرور بموقع معين وتكون هنالك عدة مسارات يفترض اختيار احدها للوصول الى الهدف فان الامر يتطلب خزن هذا الموقع قبل تركه وتجربة مسار اخر اذ يحتاج الى العودة لهذا الموقع في حالة خطا ذلك المسار . ان استخدام المكدس في مثال هذه الحالات يسمح بخزن سلسلة المواقع السابقة بحيث يمكن العودة اليها بعكس ترتيب المرور فيها.

تمرين محلول:اكتب خوارزمية لقراءة جملة STRING تنتهي بـ (.) ثم طبعها بترتيب معكوس باستخدام المكدس.

Algorithm
Being
Clear the stack
Repeat
Read a character
If character <>'.'
Then push the character onto stack
Until character ='.'
While stack is not empty do
Begin
Pop the stack
Print the character
End
End

تمرين : اعتمد خوارزمية التمرين السابق واكتب البرنامج الفرعي (procedure) لها.

const size=30;

```
const dot='.';
char stack[size];
int top;
char character;
void printreverse()
{
 clearstack();
 while(character!=dot)
   {
    cin>>character;
    if(character!=dot)
    push(character);
   }
  while(!emptystack())
    {
     pop();
     cout<<character;
    }
}
```

برنامج – ١ : تمثيل المكدس (stack) وعملياته

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define size 30
int st[size];
int item1,item;
int choice,i,top,l,m;
int fullstack( )
{
 if(top>=size)
 return(1);
 else return(0);
}
int emptystack()
{
 if(top==-1)
 return(1);
```

```
 else return(0);
}
void push(int item)
{
 top++;
 if(fullstack())
 {
  cout<<" error...the stack is full"<<endl;
  cout<<"enter any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 st[top]=item;
}
void pop()
{
 if(emptystack())
 {
  cout<<"error...the stack is empty"<<endl;
  cout<<"enter any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 {
  item=st[top];
  top--;
 }
}
void main()
{
 clrscr();
 top=-1;
 do
 {
  cout<<"representation of stack operation"<<endl;
```

```
cout<<"--------------------------------"<<endl;
cout<<"1-insertion operation(push)       "<<endl;
cout<<"2-deletion operation(pop)         "<<endl;
cout<<"3-display the content of the stack  "<<endl;
cout<<"4-exit                             "<<endl;
cout<<"select your choice  "<<endl;
cin>>choice;
switch (choice)
  {
   case(1):
   {
       cout<<"how many elements you like to enter";
       cin>>m;
       for(i=0;i<m;i++)
    {
        cout<<"enter the new element"<<endl;
         cin>>item1;
        push(item1);
       }
       break;
  }
   case(2):
   {
       cout<<"how many elements you want to delete"<<endl;
         cin>>l;
       for(i=0;i<l;i++)
       pop();
       break;
   }
   case(3):
   {
    if(top==-1)
    cout<<"error stack is empty"<<endl;
    else
    {
     cout<<"the content of the stack is:"<<endl;
        for(i=top;i>-1;i--)
```

```
        cout<<st[i];
      }
        break;
     }
    }
  }while(choice!=4);
}
```

برنامج –٢ :لقراءة جملة string وطبعها بصورة معكوسة باستخدام المكدس stack .

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define size 10
char st[size];
char item;
int top,i;
int fullstack()
{
 if(top>=size)
 return(1);
 else return(0);
}
int emptystack()
{
 if(top==-1)
 return(1);
 else return(0);
}
void push(int item)
{
 top++;
 if(fullstack())
 {
  cout<<" error...the stack is full"<<endl;
  cout<<"enter any key to exit"<<endl;
  getch();
  exit(0);
```

```
 }
 else
 st[top]=item;
}
void pop()
{
 if(emptystack())
 {
  cout<<"error...the stack is empty"<<endl;
  cout<<"enter any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 {
  item=st[top];
  top--;
 }
}
void main()
{
 clrscr();
 top=-1;
 cout<<"this program reads in any string and printed in reverse
order using stack"<<endl;
 cout<<"input your string terminated by (.)"<<endl;
 item='A';
 while(item!='.')
  {
   cin>>item);
   push(item);
  }
 top--;
 cout<<"your string in reverse order is"<<endl;
 for(i=top;i>-1;i--)
  {
   pop();
```

```
  cout<<item;
 }
getch();
}
```

برنامج –٣ : لقراءة جملة وفحص هل يمكن قراءتها من الاتجاهين اي نوعها palindrome باستخدام المكدس.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 30
char st[size],st1[size],st2[size];
char *item1,ch1,ch2;
int count,i,palindrome;
int  top1=-1,top2=-1;
int fullstack1(int *top)
{
 if(*top>=size)
 return(1);
 else return(0);
}
int emptystack1(int *top)
{
 if(*top==-1)
 return(1);
 else return(0);
}
void push(char *item,char st[size],int *top)
{
 ++(*top);
 if(fullstack1(top))
 {
```

```
  cout<<" error...the stack is full"<<endl;
  cout<<"enter any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 st[*top]=*item;
}
void pop(char *item,char st[size],int *top)
{
 if(emptystack1(top))
 {
  cout<<"error...the stack is empty"<<endl;
  cout<<"enter any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 {
 *item=st[*top];
 --(*top);
 }
}
void main()
{
 clrscr();
 palindrome=1;
 top1=-1;
 top2=-1;
 count=0;
 cout<<<"this program can reads in any string and tested if its
palindrome or  not or not (i.e it can be read from both
sides)"<<endl;
cout<<"input your string terminated by(.)"<<endl;
 ch1='A';
  while(ch1!='.')
   {
```

```
   cin>>ch1;
   push(&ch1,st1,&top1);
   count++;
  }
 count--;
 pop(&ch1,st1,&top1);
 for(i=0;i<(count/2);i++)
  {
   pop(&ch1,st1,&top1);
   push(&ch1,st2,&top2);
  }
 if((count%2)==1)
 pop(&ch1,st1,&top1);
 while (!emptystack1(&top1)&&palindrome)
  {
   pop(&ch1,st1,&top1);
   pop(&ch2,st2,&top2);
   if(ch1!=ch2)
   palindrome=0;
  }
  if(palindrome)
  cout<<"the string is palindrome"<<endl;
  else
  cout<<"the string is not palindrome"<<endl;
  getch();
}
```

٣-٣ الطابور queue

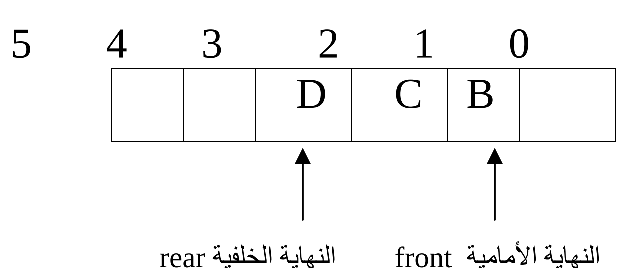
هـو هيكـل تسلسـلي (sequential) تكـون فيـه عمليـات الاضـافة فـي النهايـة الخلفيـة rear وعمليات الحذف في النهاية الاخرى (الامامية) front كما في الشكل التالي:

5	4	3	2	1	0
			C	B	A

النهاية الخلفية rear   النهاية الأمامية front

حيث نلاحظ ان العنصـر A فـي مقدمـة الطـابور يليـه العنصـر B ثم C وعنـد اضـافة عنصـر جديـد يكـون موقعـه بعـد C ، امـا عنـد حـذف عنصـر مـن الطـابور تكـون عمليـة الحـذف مـن النهايـة

الأمامية اي حذف العنصر A ويصبح الشكل اعلاه بعد اضافة العنصر D وحذف العنصر A كالاتي:



نجد أن الطابور يفيد في الفعاليات التي تتضمن جدولة الأعمال حسب ترتيب وصولها او طلبها ويمكن تلخيص هذا بالعبارة الاتية: أول من يدخل اول من يخرج FIRST IN FIRST OUT (FI FO). أي ان من يصل اولا يحصل على الخدمة اولا وتسمى عملية الإضافة الى الطابور ENQueue او Insertion اما عملية الحذف فتسمى DEQueue او Deletion .

٣-١-٣ تمثيل الطابور باستخدام المصفوفة Array Representation of queue

يطبق الطابور باستخدام مصفوفة احادية بالسعة المطلوبة (size) وبالنوع المناسب لنوع البيانات (data type) التي ستخزن فيه (int , float , الخ) مع استخدام :

- المتغير (rear) كمؤشر يشير الى موقع العنصر الاخير في الطابور.
- المتغير (front) كمؤشر يشير الى موقع العنصر الاول في الطابور

ان قيمة المؤشرين في الحالة الابتدائية (rear=-1,front=0) عندما يكون طابور خاليا(empty) من العناصر .تنفذ عملية اضافة عنصر الى الطابور بعد تحديث قيمة المؤشر (rear) ليشير الى الموقع الجديد بعد موقع اخر عنصر ، اما عند تنفيذ عملية حذف عنصر من الطابور فيحدث المؤشر (front) ليشير الى موقع العنصر التالي بعد حذف العنصر في المقدمة،ولنفترض ان لدينا الطابورQ سعته ٦عناصر وننفذ عليه سلسلة العمليات الاتية:

الحالة	المؤشر		المؤشر		q[1]	q[2]	q[3]	q[4]	q[0]
q[5]									
	F			R					
- الطابور خالي	0	-1	-	-	-	-	-	-	
- اضافة العنصرA	0		0	A	-	-	-	-	-
- اضافة العنصر B	0	1	A	B	-	-	-	-	
- اضافة العنصرC	0	2	A	B	C	-	-	-	
- حذف عنصر	1			2	-	B	C	-	-
-									
- اضافة العنصرD	1	3	-	B	C	D	-	-	
- اضافة العنصر E	1	4	-	B	C	D	E	-	
- حذف عنصر	2		4	-	-	C	D	E	-
- حذف عنصر	3			4	-	-	-	D	E
-									

ويعرف الطابور برمجياً باستخدام العبارات البرمجية التالية:

عملية الاضافة للطابور ADD TO QUEUE :تعتمد الخطوات الاتية لاضافة عنصر واحد الى الطابور:

١- التحقق بان الطابور غير مملوء (NOT FULL) اي ان المؤشر (rear!=SIZE-1) لتجنب حالة الفيض (OVER FLOW) وتعذر تنفيذ عملية الاضافة عند ذلك .

٢- تحديث قيمة المؤشر (rear=rear+1) ليشير الى الموقع التالي.

٣- اضافة العنصر الجديد في الموقع الجديد QUEUE [rear]

عملية الحذف من الطابور DELETE FROM QUEUE :نعتمد الخطوات الاتية لحذف عنصر واحد من الطابور:

١- التحقق بان الطابور غير خال (NOT EMPTY) اي ان المؤشر (rear>=front) لتجنب حالة الغيض (UNDER FLOW) وتعذر تنفيذ عملية الحذف.

٢- اخذ العنصر من الموقع الذي يشيراليه المؤشر(front) وخزنه وقتيا في متغير مستقل وليكن ITEM=QUEUE[front]

٣- تحديث قيمة المؤشر (front=front+1) ليشير الى موقع العنصر الاتي للعنصر الذي تم حذفه.

ملاحظة: هنا ايضا يتضح ان الخطوتين٣(٢) في عملية الحذف معكوسة الترتيب عنها في عملية الاضافة.

٣-٣-٢ خوارزميات الطابور Queue's Algorithms

ادناه مجموعة من الخوارزميات لتغطية العمليات التي تنفذ على الطابور:

١- خوارزمية الاضافة Add Queue

```
If      queue is full
Then over flow ←        true
Else
      Over flow ←       false
      Rear ←       rear+1
       Queue (rear) ←        new element
```

٢-خوارزمية الحذف delete queue

```
If      queue is empty
Then under flow ←        true
Else
       Under flow ←          false
        Element ←               queue(front)
        Front  ←              front +1
```

٣-خوارزمية ملئ الطابور full queue

هذه الخوارزمية للتحقق من الطابور ان كان مملوء ام لا اعتمادا على قيمة المؤشر(rear) قبل عمليات الاضافة.

```
If            rear= size-1
Then          fullqueue ←                true
```

Else fullqueue ← false

٤- خوارزمية خلو الطابورempty queue

هذه الخوارزمية للتحقق من الطابور ان كان خاليا ام لا اعتمادا على قيمة الؤشر (front) قبل عملية الحذف.

```
If        rear<front
Then           emptyqueue ←           true
Else      empty queue ←          false
```

٥- خوارزمية إخلاء (تفريغ) الطابور Clear Queue

هذه الخوارزمية تستخدم لغرض تهيئة الطابور واخلائة من العناصر بجعل قيمة كل من المؤشرين (front =0 ,rear=-1)

```
Fornt ← 0
Rear ← -1
```

٣-٣-٣ البرامج الفرعية لتنفيذ عمليات الطابور

فيما يلي مجموعة من البرامج الفرعية (Functions , Procedures)أعدت بنفس أسلوب البرامج الفرعية للمكدس مع افتراض وجود التعريف التالي للمتغيرات في مقدمة البرنامج .

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define size 10
int q[size];
int item;
int front,rear;
```

١- برنامج فرعي لاخلاء الطابور

```
void clearqueue()
{
 rear=-1;
 front=0;
}
```

نلاحظ عدم الحاجة للمرور على جميع مواقع المصفوفة والاكتفاء فقط بجعل قيمة المؤشر rear مساويا 1- و المؤشر front مساويا 0 للصفر .(يستدعى هذا البرنامج الفرعي في البداية لجعل الطابور خالياً) .

٢- برنامج فرعي للتحقق من امتلاء الطابور

```
int fullqueue()
{
 if(rear>=size)
 return(1);
```

```
 else return(0);
}
```

هذه الدالة (function) تقوم بارجاع قيمة(1) اي true عندما يكون الطابور مملوء وتقوم بارجاع قيمة(0) اي false عندما يكون الطابورغير مملوء. (يستدعى هذا البرنامج الفرعي داخل البرنامج الفرعي (procedure insert Queue)الذي ينفذ عملية الاضافة).

٣-برنامج فرعي للتحقق من خلو الطابور

```
int emptyqueue()
{
 if(rear<front)
 return(1);
 else return(0);
}
```

هذه الدالة (function) تقوم بارجاع قيمة (1) اي true عندما يكون الطابور فارغا وتقوم بارجاع قيمة
(0) اي false عندما يكون الطابورغير فارغ. (يستدعي هذا البرنامج الفرعي داخل البرنامج الفرعي (Procedure Delete Queue)الذي ينفذ عملية الحذف) .

٤- برنامج فرعي لاضافة عنصر واحد الى الطابور

```
void addqueue(int item)
{
 rear++;
 if(fullqueue())
 {
  cout<<"error queue is full"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
     q[rear]=item;
}
```

ملاحظة : يمكن استدعاء هذا البرنامج الفرعي داخل البرنامج الرئيسي (main Program)بأي عدد من المرات باستخدام أحد ايعازات التكرار مثل (for,do_while) بقدر عدد العناصر المطلوب أضافتها .

٥- برنامج فرعي لحذف عنصر واحد من الطابور

```
void deleteq()
{
 if(emptyqueue())
```

```
{
 cout<<"error queue is empty"<<endl;
 cout<<"press any key to exit"<<endl;
 getch();
 exit(0);
}
else
{
 item=q[front];
 front++;
}
}
```

ملاحظة : يمكن استدعاء هذا البرنامج الفرعي من قبل البرنامج الرئيسي بأي عدد من المرات بقدر عدد العناصر المطلوب حذفها باستخدام أحد ايعازات التكرار المشار اليها.

٣-٣-٤ تمثيل الطابور باستخدام القيد

يستخدم القيد في تمثيل الطابور والمؤشرين (front ,rear)في هيكل بياني واحد أي أن القيد يتكون من ثلاثة أجزاء ، الجزء الأول يمثل مصفوفة الطابور والجزء الثاني وهو حقل يمثل المؤشر (Front)والجزء الثالث هو حقل أخر يمثل المؤشر (rear).

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define size 8
char item;
struct queue
{
  char elements[size];
  int rear;
  int front;
 }q;
```

Queue:

A	B	C	D					**0**	**3**

elements   Front   rear

لاضافة عنصر جديد لهذا الطابور نتبع الخطوات التالية :-

١- نحذف قيمة المؤشر (Rear)الذي هو حقل في القيد Queue ليصبح ٤

```
q .rear= q. rear+1;
```

٢- نضيف العنصر الجديد (E) في الموقع الجديد ٤

```
q.elements [q.raer]=E;
```

وبهذا الطابور بالصورة التالية:

A	B	C	D	E				**0**	**4**

elements   front   rear

q.elements[4]

اما عند حذف عنصر من الطابور فنتبع الخطوات الاتية:

١- ناخذ العنصر من مقدمة الطابور كما يشير اليه المؤشر (front) بالايعاز:

Item =q. elements (q.front)

٢- تحديث قيمة المؤشر (front) من 0 ليصبح 1 بالايعاز :

q.front=q.front+1

ويصبح الطابور بالصورة الاتية:

	B	C	D	E				1	4

elements   front   rear

<u>تمرين</u> : اعد كتابة البرامج الفرعية لعمليات الطابور باستخدام القيد.

٣-٣-٥ تطبيقات الطابور:

من التطبيقات الشائعة للطابور في مجال الحاسوب:

* جدولة الاعمال (job scheduling) ففي نظام معالجة الدفعات (batch processing) تنظم الاعمال المطلوب تنفيذها في طابور حسب وقت وصولها ومن ثم تنفذ بالتتابع واحدا بعد الاخر.
* كما تستخدم انظمة التشغيل الطابور في جدولة استخدام المصادر المختلفة للحاسوب (resources) فمثلا طابور للاعمال التي تحتاج وقت للاخراج على الطابعة واخر للادخال ، واستخدام القرص ، وهكذا باقي الاجهزة.

<u>تمرين</u> : اعد كتابة البرنامج الفرعي (Add Queue)لاضافة عنصر الى الطابور باستخدام القيد

```
void addqueue()
{
 if(q.rear>=size-1)
 {
   cout<<"error…queue is full"<<endl;
   cout<<"press any key to exit"<<endl;
   getch();
   exit();
  }
  else
  {
   q.rear++;
   q.elements[q.rear]=item;
   }
 }
```

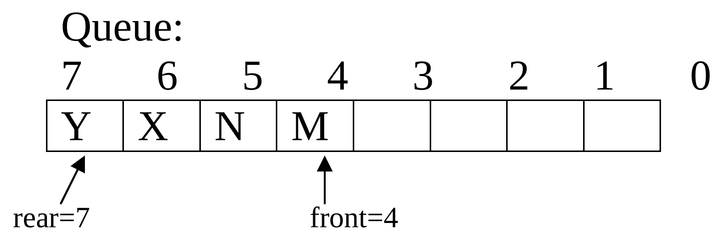
تمرين: اكتب برنامجا فرعيا لاضافة ٥ عناصر الى الطابور (LINE) الذي سعته ٢٥ عدد صحيح.

```
const size=25;
int line[size];
void addqueue()
{
  int i,item;
  for(i=0;i<5;++i)
    {
     if(rear>=size-1)
      {
        cout<<"error…queue is full"<<endl;
        cout<<"press any key to exit"<<endl;
        getch();
        exit(0);
        }
        else
          {
           rear++;
          cin>>item;
          line[rear]=item;
          }
        }
  }
```

تمرين : اكتب برنامجا فرعيا لقراءة جملة string تتكون من جزأين (two substring) يفصلها الرمز ،.، ثم التحقق من كونهما متطابقتان ام لا باستخدام الطابور.

٣-٤ الطابور الدائري circular queue

لاحظنا انه عند اضافة عنصر الى الطابور يتطلب تدقيق (فحص) قيمة المؤشر (rear=size-1) اذ ان ذلك يعني ان الطابور مملوء حتى وان كانت هنالك مواقع خالية في مقدمة الطابور كما في الشكل الاتي:



أي أننـا سنخسـر مسـاحة خزنيـة دون اسـتخدام ، ولتجنـب هـذه الحالـة نسـتطيع اسـتخدام الطـابور استخداما دائريا وذلك بـان نسمح للمؤشر (rear) بالـدوران الـى النهايـة الاماميـة للطـابور (-wrap around) عندما يكون هنالك مواقع خالية فيها.

ان خصائص هذا الطابور هي كاتي :

١- ان المؤشر front يشير الى الموقع الذي امام اول عنصر في الطابور.

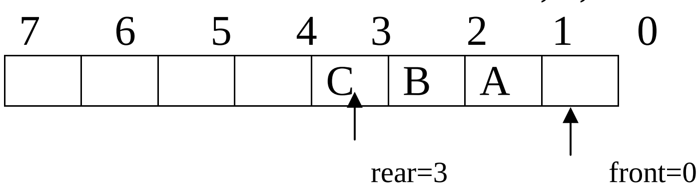
٢- أن المؤشر rear يشير الى موقع العنصر الاخير في الطابور.

٣- عندما يصل المؤشر rear الى الموقع الاخير في الطابور اي rear=size-1 نجعله يدور الى البداية اي rear=0 وكذلك الحال بالنسبة للمؤشر front.

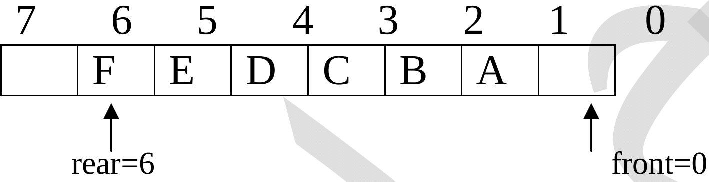
٤- ان اكبر عدد من العناصر التي يستوعبها هذا الطابور هو size-1 لان المؤشر front يشير الى الموقع الخالي امام اول عنصر في الطابور.

ولنأخذ هذه الأمثلة التوضيحية الآتية لطابور دائري سعته (8):

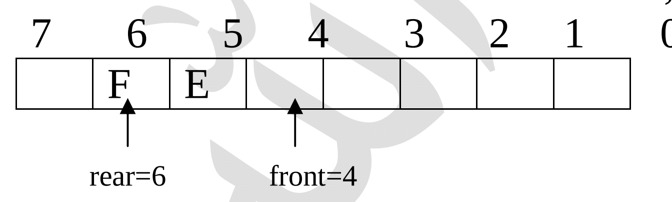
الحالة الأولى : يحتوي الطابور ثلاثة عناصر C,B,A



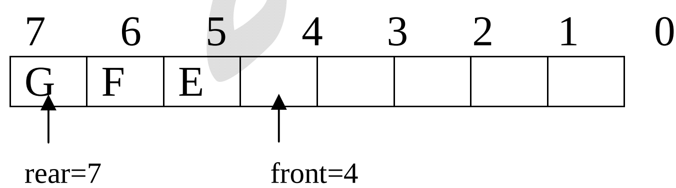
الحالة الثانية : بعد اضافة العناصر F,E,D



الحالة الثالثة : بعد حذف العناصر D,C,B,A

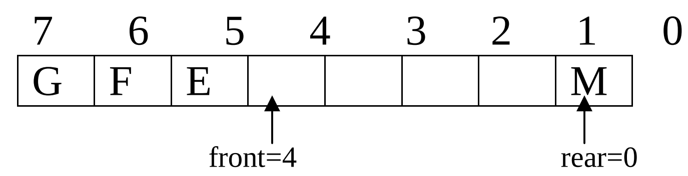


الحالة الرابعة: بعد اضافة العنصر G



لاحظ هنـا في الطـابور البسيط يعتبـر الطـابور مملـوء لان rear=SIZE-1=7 مع انـه يوجد مواقع خالية في المقدمة اما في الطابور الدائري فنستطيع اضافة عنصر كما في الحالة التالية.

الحالة الخامسة :لاضـافة العنصـرM لاحظ دوران المؤشر rear الـى الجهـة الاماميـة بعد وصـوله الـى الموقع الاخير في الطابور .



الحالـة السادسـة : بعد اضـافة العناصـر x,p,n لاحظ هنـا ان الطـابور اصبح مملـوءا لاننـا اذا اردنـا اضـافة عنصـر فيجـب تحـديث قيمـة المؤشـر rear لصـبح مسـاويا 4 ويكـون مساويا لقيمـة المؤشـر front=4 الذي يجب ان يبقى موقعه خاليا كما ان سعة الطابور هي size-1 اي ان 7=1-8 وهذا ما يحتويه حاليا ، لذا فيتعذر الاضافة في هذه الحالة.

7	6	5	4	3	2	1	0
G	F	E		X	P	N	M

front=4 rear=3

برنامج فرعي لاضافة عنصر الى الطابور الدائري

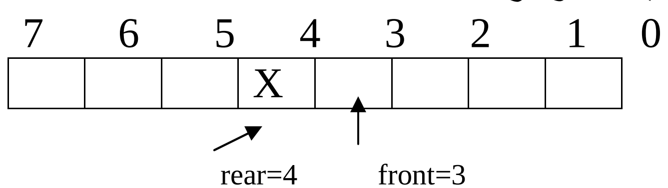
```
void addcq(int item)
{
 if(rear>=size-1)
 rear=0;
 else
 rear++;
 if(rear==front)
 {
  cout<<"error...circular queue is full"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
cq[rear]=item;
}
```

برنامج فرعي لحذف عنصر من الطابور الدائري

```
void deletecq()
{
 if(rear==front)
 {
  cout<<"error...circular queue is empty"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 {
  if(front>=size-1)
```

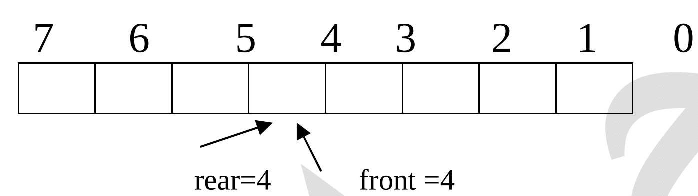
```
  front=0;
  else front++;
  item=cq[front];
 }
}
```

توضيح : لناخذ طابورا دائريا سعته ٨ عناصر يحتوي حاليا على عنصر واحد هو x كما في الرسم التوضيحي ادناه حيث المؤشر front=3 وقيمة المؤشر rear=4 .



عند حذف عنصر واحد ماذا سيحصل :

- نحدث قيمة المؤشر front:=front+1 اي يصبح front =4
- ناخذ العنصر من موقعه item =q [front]
- يصبح شكل الطابور الدائري كالاتي:



rear=front=4 والطابور خال من العناصر لذا فان اي عملية حذف لاحقة كما في البرنامج الفرعي deleteCQ يتعذر تنفيذها.

ملاحظة مهمة: نجد ان الشرط نفسه rear==front يستخدم في البرنامج الفرعي ADDCQ ليعني ان الطابور مملوء وفي البرنامج الفرعي DELETECQ ليعني ان الطابور خال وهذا لا يعني التناقض لان تسلسل الخطوات يختلف اذ يرد عند الاضافة بعد تحديث قيمة المؤشر rear ويرد عند الحذف في البداية.

٣-٥ الطابور الثنائي (المزدوج)(DOUBL ENDED QUEUE(DEQEUE)):

هو هيكل تسلسلي يمكن اضافة او حذف العنصر من اي من طرفية ويمثل بمصفوفة احادية مع اربعة مؤشرات هي:

F1 يشير الى موقع اول عنصر في الطابور عند استخدامه من الجهة اليمنى.
R1 يشير الى موقع اخر عنصر في الطابور عند استخدامه من الجهة اليمنى.
F2 يشير الى موقع اول عنصر في الطابور عند استخدامه من الجهة اليسرى.
R2 يشير الى موقع اخر عنصر في الطابور عند استخدامه من الجهة اليسرى.

كما في الشكل التالي

F2

R2

7	6	5	4	3	2	1	0
		T	N	M	X		

R1 F1

اذ يمكن حذف العنصر X باستخدام المؤشر F1 واضافة العنصر T باستخدام المؤشر R1 ، كما يمكن حذف العنصر T باستخدام المؤشر F2 واضافة عنصر بعد العنصر X باستخدام المؤشرR2 .

<u>برنامج ٤- تمثيل الطابور QUEUE وعملياته</u>

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define size 10
int q[size];
int item;
int front,rear,choice,i,l,m;
int fullqueue()
{
 if(rear>=size)
 return(1);
 else return(0);
}
int emptyqueue()
{
 if(rear<front)
 return(1);
 else return(0);
}
void addqueue(int item)
{
 rear++;
 if(fullqueue())
 {
  cout<<"error queue is full"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 q[rear]=item;
```

```
 }
void deleteq()
{
 if(emptyqueue())
 {
  cout<<"error queue is empty"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 {
  item=q[front];
  front++;
 }
}
void main()
{
 clrscr();
 rear=-1;
 front=0;
 do
 {
  cout<<"representation of queue operation"<<endl;
  cout<<"--------------------------------"<<endl;
  cout<<"1-insertion operation      "<<endl;
  cout<<"2-deletion operation        "<<endl;
  cout<<"3-display the content of the queue"<<endl;
  cout<<"4-exit                              "<<endl;
  cout<<"select your choice"<<endl;
  cin>>choice;
  switch (choice)
    {
     case(1):
     {
         cout<<endl<<"how many elements you like to enter";
         cin>>m;
```

```
        for(i=0;i<m;i++)
        {
           cout<<"enter the new element"<<endl;
           cin>>item;
        addqueue(item);
           cout<<endl<<"rear="<<rear<<",front="<<front";
        }
        break;
    }
     case(2):
      {
          cout<<"how many elements you want to delete"<<endl;
          cin>>l;
          for(i=0;i<l;i++)
          deleteq();
          break;
         }
     case(3):
      {
       if(rear<front)
          cout<<"error queue is empty"<<endl;
          else
       {
          cout<<"the content of the queue is:"<<endl;
          for(i=rear;i>=front;i--)
          cout<<endl<<q[i]<<endl;
         }
        break;
      }
     }
   }while(choice!=4);
}
```

برنامج -٥ تمثيل الطابور الدائري (CIRCULAR QUEUE) وعملياته

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
```

```
#define size 6
int cq[size];
int front,rear,choice,i,l,m;
int item;
void addcq(int item)
{
 if(rear>=size-1)
 rear=0;
 else
 rear++;
 if(rear==front)
 {
  cout<<"error...circular queue is full"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 cq[rear]=item;
}
void deletecq()
{
 if(rear==front)
 {
  cout<<"error...circular queue is empty"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 {
  front++;
  if(front>=size-1)
  front=0;
  else
  item=cq[front];
  cq[front]=0;
```

```
 }
}
void main()
{
 clrscr();
 rear=0;
 front=0;
 for(i=0;i<size;i++)
 cq[i]=0;
 do
 {
  cout<<"representation of queue operation"<<endl;
  cout<<"--------------------------------"<<endl;
  cout<<"1-insertion operation      "<<endl;
  cout<<"2-deletion operation       "<<endl;
  cout<<"3-display the content of the circular queue"<<endl;
  cout<<"4-exit                               "<<endl;
  cout<<"select your choice"<<endl;
  cin>>choice;
  switch (choice)
    {
     case(1):
     {
        cout<<"how many elements you like to enter"<<endl;
        cin>>m;
        for(i=0;i<m;i++)
       {
          cout<<"enter the new element"<<endl;
          cin>>item;
          addcq(item);
           cout<<"rear="<<rear<<"  front="<<front;
         }
        break;
     }
        case(2):
       {
           cout<<"how many elements you want to delete"<<endl;
```

```
            cin>>l;
            for(i=0;i<l;i++)
            deletecq();
            break;
           }
          case(3):
         {
            cout<<"the content of the circular queue is:"<<endl;
            for(i=0;i<size;i++)
            cout<<endl<<cq[i]<<endl;
            cout<<"rear="<<rear<<"  front="<<front;
            break;
           }
          }
      }while(choice!=4);
   }
```


أسئلة الفصل

1- Let s be a stack of size 15 character, write a procedure push to add one element.

2- Write a procedure to pop three elements from the stack TABLE of size 20 integers.

3- Let s for stacking an element in the stack, and U for un stacking.
   If the order of stack input stream is 1 2 3 4 5
   a- what is the output if we execute the following operations SSUUSSSUUU
   b- Which of the following permutations can be obtained as output stream (explain the reason for each case).

i- 51324
ii- 23514
iii- 32154

4- if the stack input stream is A B C D E F what is the sequence of operations to get the out put C B D E F A?

5- write a function to check if the stack St is empty.

6- Explain how we can use the record for stack implementation?

7- State the main applications of the stack?

8- Explain how we can use stack to achieve the processing of procedures calls in computer programming?

9- Consider the following program structure

```
Begin   (main program)
….
….
….
Call A
….
….
Call B
….
….
…. Call X  ….
….          ….
….          ….
….          ….
….          Call M
….          ….
….          ….
….          Call N
….          ….
….          ….
….    Call Y
….    ….
….    ….
Call C
….
….
```

END

Explain in details how we can use the stack in processing of such program.

10 – Convert the following infix expressions into postfix notations using two stacks:

i- a+b^2/4-(c*5/8-f) ^3

ii- m^3 or n-b/2 and (m+n)

iii- a+b+c*(-d/3^4)and not f

iv- a^(b/2)or(x+y/3-w)and (c^2)^3

v- x^n+(m-p*4)^f and (-b)^2/c+6

vi- a+b-c or (x*2^n)-m/p and f^2

vii- a^(b/2)and (x-y/3+w)or(c^3)^2

viii- n^x-(p*4+m)/f or –c/b^2

ix- b-a+c and n^x-(p/m or f^2)

11-Execute the following postfix notation using the stack ab*cde^/+ when a=5, b=6,c=8,d=2,e=2

12- Execute the following infix notation using the stack

a/(b*c/2)^4+m   if a=10,b=8,c=4,m=20

13- Write an algorithm that reads in a string and print its characters in reverse order (note: the string terminator is a .، Which should not be printed as a part of the reversed string).

14- Write a procedure that that reads in a string and print its characters in reverse order (note: the string terminator is a blank Which should not be printed as a part of the reversed string).

15- Write a procedure to add an element to the queue called (LINE) of size 30 characters.

16- Write a procedure to delete an element from the circular queue called (BOB)of size 15 integers.

17- What are the main differences between simple queue and circular queue? Explain that in detail.

18- What are the applications of the queue in the computer?

الفصل الرابع

٤-١-١ التخصيص الخزني التسلسلي sequential allocation of storage

ان ابسط الطرق لخزن القائمة الخطية هو استخدام الخزن التسلسلي في ذاكرة الحاسوب اي يتم الخزن في مواقع متتابعة (متسلسلة) ويمكن ان نعرف موقع اي عنصر اذا عرفنا موقع العنصر الاول الذي هو عنوان البداية base address ومواقع العناصر التالية ستكون منسوبة له. فالعنصر k سيكون موقعه تاليا لموقع العنصر k-1 وهكذا بقية العناصر.

المزايا advantages:

١- اكثر سهولة في التمثيل والتطبيق .
٢- يكون اقتصاديا اكثر لانه يستخدم مساحة خزنية اقل.
٣- اكثر كفاءة في الوصول العشوائي.
٤- مناسب جدا عند التعامل مع المكدس.

المساوئ:disadvantage:

١- صعوبة تنفيذ عمليات الاضافة والحذف.
٢- يتطلب التعريف المسبق وتحديد عدد العناصر المطلوب خزنها.

٤-١-٢ التخصيص الخزني الديناميكي:dynamic allocation of storage:

ان الطريقة الاخرى للخزن هي استخدام رابط link او مؤشر مع كل عنصر يحتوي عنوان موقع العنصر التالي لذلك لا توجد ضرورة لخزن البيانات في مواقع متعاقبة (متسلسلة) بل يمكن خزن اي عنصر بياني في اي موقع ، ولهذا فكل عنصر (عقدة) يتكون من جزاين هما:

١- الجزء الاول : يحتوي البيانات data
٢- الجزء الثاني : حقل يحتوي على عنوان موقع العنصر التالي link

X:

(x)link	(x)data

فالعنصر x: يتكون من الجزاين (x) link و data(x)

المزايا : سهولة تنفيذ عمليات الاضافة والحذف لأي عنصر اذ لايتطلب اكثر من تحديث قيمة حقل المؤشر الذي يعطي عنوان الموقع التالي.

المساوئ: يحتاج الى مساحة خزنية اكبر لتمثيل حقل المؤشر اضافة الى البيانات الاساسية.

٤-١-٣ المقارنة بين الخزن التسلسلي والخزن الديناميكي

يمكن ان تتركز المقارنة في النقاط التالية:
أ- المساحة الخزنية Amount of storage:
ان اسلوب الخزن الديناميكي يحتاج الى مساحة خزنية اكبر لان كل عنصر في الهيكل البياني يحتاج الى مؤشر (اي موقع خزن اضافي) يحتوي عنوان موقع العنصر التالي ، وهكذا لجميع العناصر.
ب- عمليات الاضافة والحذف:
ان هذه العمليات اسهل تنفيذا في الخزن الديناميكي لانها تتطلب غير تغيير قيمة المؤشر ليحتوي عنوان موقع موقع العنصر بعد الاضافة او الحذف . اما في الخزن التسلسلي فانه يتطلب تزحيف shifting العناصر.
ج- الوصول العشوائي للعناصر Random Access:
ان اسلوب الخزن التسلسلي يعتبر افضل في الوصول عشوائيا لاي عنصر من عناصر الهيكل البياني مباشرة (k-th element from the start) اما في الخزن الديناميكي فانه يتطلب البدء من اول عنصر ثم العناصر التالية با لتتابع لحين الوصول الى موقع خزن العنصر المطلوب.
د- الدمج والفصل MERG AND SPLIT
في الخزن الديناميكي تكون هذه العمليات اسهل تنفيذا وذلك بتغيير قيمة المؤشر للعناصر للعقد في مواقع المج او الفصل اما في الخزن التسلسلي فالعمل اكثر تعقيدا اذ قد يتطلب تزحيف العناصر واعدة تنظيم الخزن Reorganization of storage

٤-٣ المؤشرات pointers

لكي نستطيع فهم تمثيل وعمل الهياكل البيانية باستخدام الخزن الديناميكي لابد من شرح كيف يمكن تعريف المؤشرات وطريقة استخدامها برمجيا. عند التعامل مع المصفوفات arrays سبق ان استخدمنا الدليل index للوصول الى موقع احد عناصرها ، اي ان الدليل عبارة عن متغير استخدامه البرنامج في الوصول الى الموقع المطلوب ويستخدمه المترجم compiler كعنوان لموقع معين في الذاكرة بنفس الاسم (اسم الدليل).
ان هذا المؤشر (الدليل) يعتبر مؤشرا نسبيا relative اي يدلنا على موقع العنصر بالنسبة الى موقع بداية base address خزن المصفوفة في الذاكرة هنالك حالات تستوجب بناء هياكل بيانية مختلفة السعة الخزنية خلال مرحلة تنفيذ البرنامج وهي الهياكل البيانية الموصولة اذ يكون لكل عنصر بياني فيها حقل (جزء) اضافي link يستعمل كمؤشر يشير الى موقع العنصر التالي ، اي ان هذه الهياكل تكون تنمو وتضيف عناصر جديدة لها بصورة ديناميكية خلال مرحلة تنفيذ البرنامج عند الحاجة لمواقع جديدة.
ان لغة ++C توفر اسلوب استخدام المؤشر pointer للتعامل مع الهياكل البيانية الموصولة . فالمؤشر هو عبارة عن نوع من البيانات (data type) قيمته تمثل عنوان موقع عنصر عقدة في الذاكرة.
لناخذ الهيكل البياني الموصول التالي:

P → [Data | Link] (A) → [Data | Link] (B) → [Data | Link] (C) ⏚

NULL

لان كل عنصر يتكون من جزاين فيعرف كقيد Record وفق الاتي:

```
#include<iostream.h>
#include<conio.h>
struct node{
            int data;
            struct  node*link;
  };
```

هذا التعريف يعني ان القيد اسمه (node) يتكون من جزأين:

<u>الجزء الأول</u>:Data ونوعه integer ويمكن تعريفه بأي نوع آخر.

<u>الجزء الثاني</u>:Link ونوعه pointer اي نفس النوع المبين في المقدمة باعتباره مؤشرا الى العنصر node.

يضاف الى التعريف أعلاه لنوع البيانات المؤشرات التي تستخدم في البرنامج الرئيسي و هي كالآتي:

```
struct node *p, *q, *r, *start;
```

<u>مثال</u>: لتعريف هيكل بياني موصول تتكون عناصره البيانية من جزاين هما (اسم الطالب - عمره) .

```
#include<iostream.h>
#include<conio.h>
struct student{
               int age;
               char name[10];
               struct student*link;
               }*first;
```

هنا تعريف للمؤشر الرئيسي و الذي يشير الى بداية الهيكل البياني ويظهر الهيكل البياني بالرسم كالآتي:

	العنصر الأول				العنصر الأخير		
first →	name	age	link →	name	age	link	→ NULL

NULL

لاحظ ان مؤشر العنصر الأخير قيمته NULL ويعني لا شيء بعده.

وتوفر لغة C لبناء الهيكل البياني بالإضافة للمؤشرات ما يلي:

<u>أ-البرنامج الفرعي new</u>:

إذا كان لدينا مؤشر معرف مسبقا مثل ;int *p فيستخدم البرنامج الفرعي بالصيغة

```
P=new int;
```

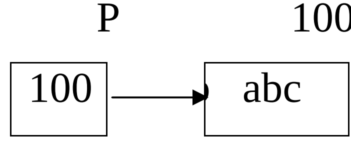
ليعني حجز موقع في الذاكرة والمؤشر p يشير اليه.

ان قيم المؤشرات في لغة C تسمح باستخدام عبارات الإحلال(=)والمقارنة بينها(==) وكذلك استخدامها كمعالم (parameters) في البرامج الفرعية(Functions & Procedures).

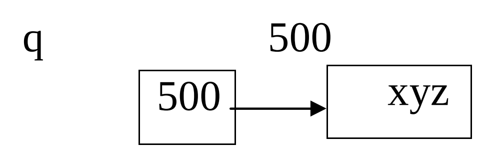
مثال: لنأخذ التعريف التالي :

```
int *p, *q;
```

ويتضمن تعريف مؤشرين p,q إذ يعني كل منهما موقع محتوياته هي قيمة عددية تمثل عنوان موقع في الذاكرة (memory address) ولنفرض انهما يشيران الى الموقعين التاليين:

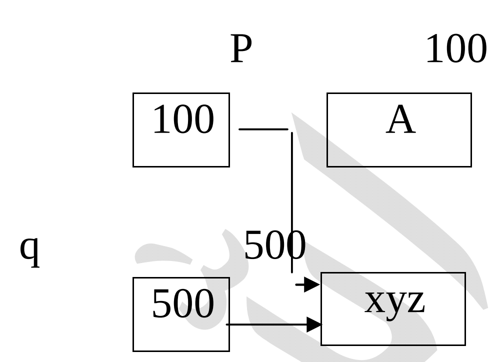


حيث الموقع p يحتوي القيمة 100 التي هي عنوان موقع في الذاكرة محتوياته abc.



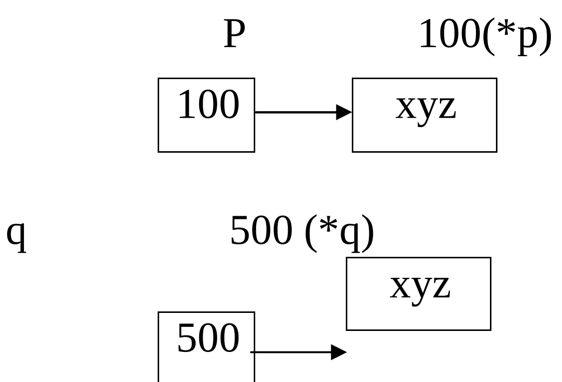
اما الموقع q فيحتوي القيمة 500 التي هي عنوان موقع في الذاكرة محتوياتهxyz.

فلو استخدمنا عبارة الإحلال (assignment) بالصيغة التالية {p=q; }إذ تعني ان قيمة p ستأخذ قيمة q وتصبح كلاهما p=500,q=500 اي ان المؤشرين p,q سيشيران الى نفس الموقع (500) كما في الرسم:



اما إذا استخدمنا الصيغة (*p= *q;) حيث p* تعني محتويات الموقع الذي يشير اليه المؤشر اي ان محتوياته هي abc اما q* تعني محتويات الموقع الذي يشير اليه المؤشر q اي ان محتوياته هي xyz.

لذا فالصيغة أعلاه تعني نسخ محتويات الموقع q* الى الموقع p* ليصبح محتوى الموقعين(xyz) كما في الرسم.



ب- البرنامج الفرعي delete( ):

توفر لغة ++C هذا البرنامج الفرعي ووظيفته هو تحرير الموقع الذي حجز باستخدام delete( ) ويكتب بالصيغة التالية: (delete(p);) ، ليعني تحرير موقع الذاكرة الذي يشير اليه المؤشر (p) اي ان البرنامج لا يحتاج الى استخدامه وهذا يحقق الخاصية المهمة للخزن الديناميكي الذي يقلل استخدام المساحة الخزنية بقدر الحاجة الفعلية فيسمح بخلق(حجز) الموقع عند الحاجة اليه وحذفه او تحريره عند انتهاء استخدامه. لذا فعند تنفيذ عملية حذف اي عنصر من الهيكل الموصول يستخدم(delete( )) بعده مباشرة.

٤-٣ القائمة الموصولة linked list

هي مجموعة من العناصر (العقد) التي كل منها يحتوي البيانات data والمؤشر link الذي يشير الى العنصر (العقدة) التالي في القائمة.

فالعنصر x يتكون من الجزأين link,data

X:

Data(x)	Link(x)

مثال:

لنأخذ قائمة موصولة تتكون من أربعة عناصر كالآتي:-

300 500 100 700

start ⟶ | A | 00 | ⟶ | B | 100 | ⟶ | C | 700 | ⟶ | D | |

data link data link data link data link

NULL

نلاحظ ما يلي:

start: هو المؤشر الرئيسي الذي يشير الى بداية اول عنصر في القائمة و ستكون قيمته في هذا المثال(300).

محتويات حقل المؤشر للعنصر الأول هو(500) ويدل على موقع العنصر الثان:(500 (link(A)=

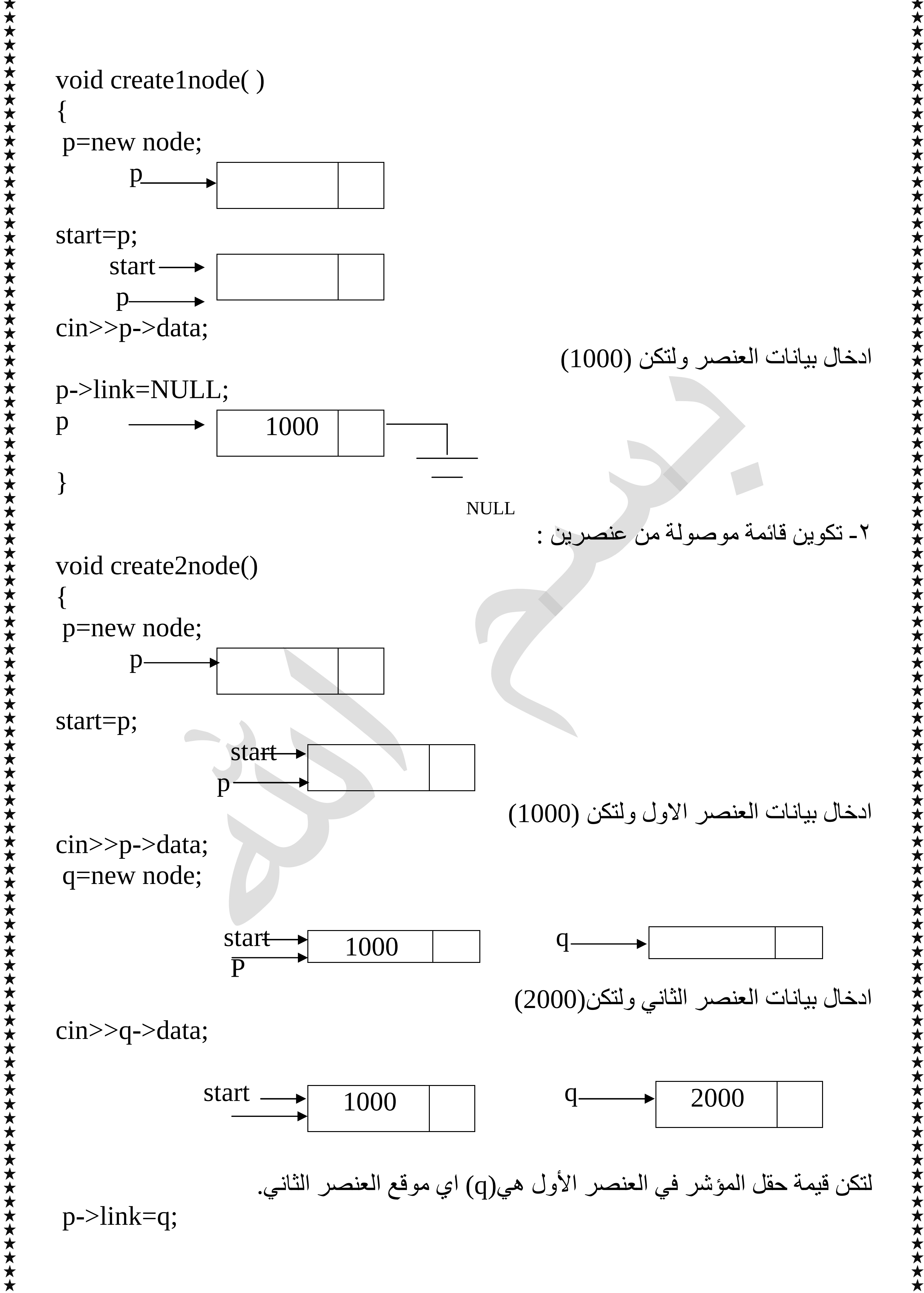
محتويات حقل المؤشر للعنصر الثاني هو(100) ويدل على موقع العنصر الثالث:(100 (link(B)=

محتويات حقل المؤشر للعنصر الثالث هو(700) ويدل على موقع العنصر الرابع:(700 (link(C)=

محتويات حقل المؤشر للعنصر الرابع هو(NULL)اي لا يوجد عنصر بعد العنصر الرابع:((link(D)=Null

فيما يأتي بعض الصيغ البرمجية (مقطع من برنامج) (Segment of Code) لتوضيح كيفية تكوين هيكل موصول او تنفيذ بعض العمليات كالإضافة والحذف مع استخدام التعريف السابق.

١- تكوين قائمة موصولة من عنصر واحد (create a linked list (of one node



```
void create1node( )
{
 p=new node;
```

```
start=p;
```

```
cin>>p->data;
```

ادخال بيانات العنصر ولتكن (1000)

```
p->link=NULL;
```

```
}
```


٢- تكوين قائمة موصولة من عنصرين :

```
void create2node()
{
 p=new node;
```

```
start=p;
```

ادخال بيانات العنصر الاول ولتكن (1000)

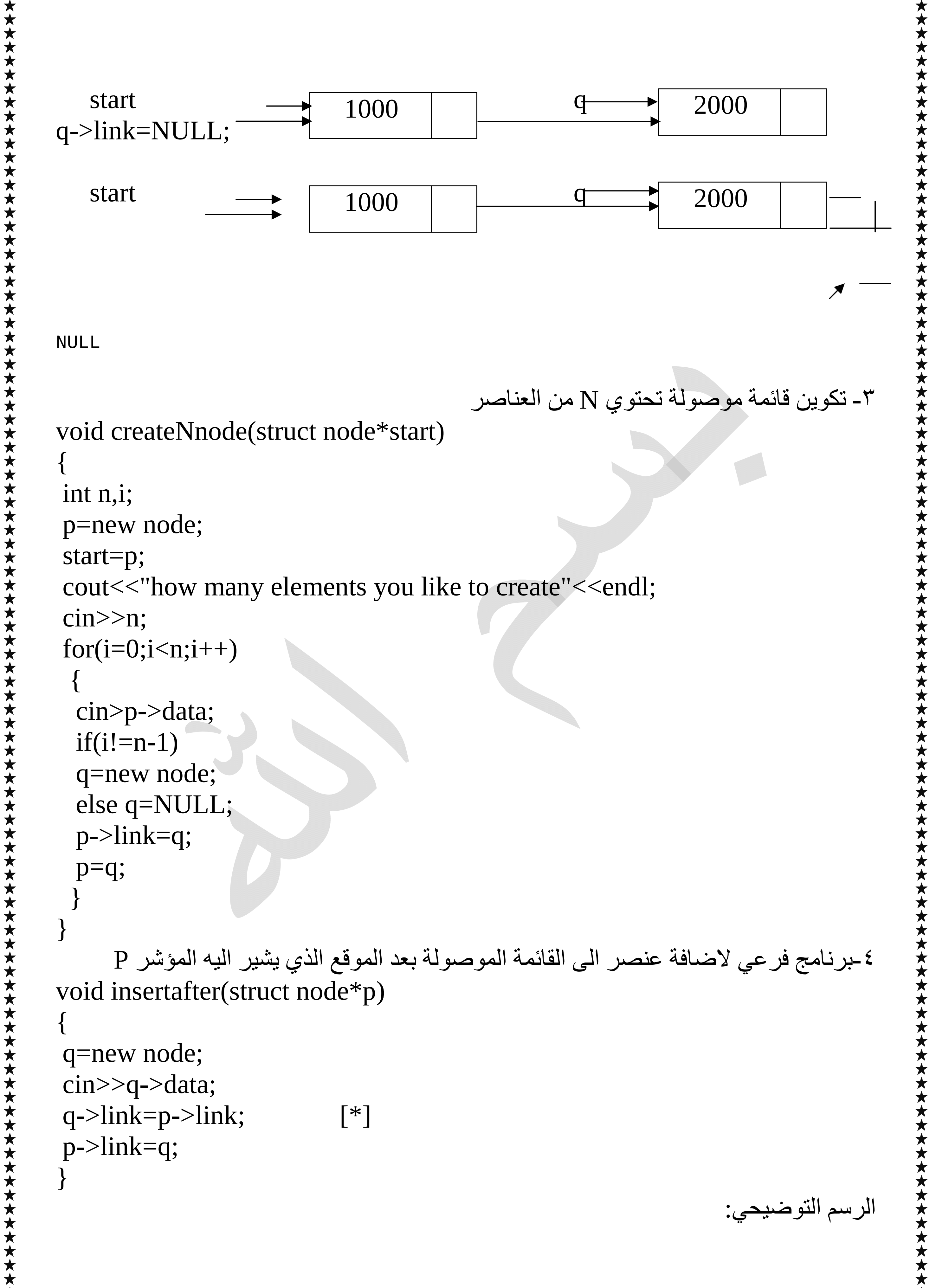
```
cin>>p->data;
 q=new node;
```

ادخال بيانات العنصر الثاني ولتكن(2000)

```
cin>>q->data;
```

لتكن قيمة حقل المؤشر في العنصر الأول هي(q) اي موقع العنصر الثاني.

```
 p->link=q;
```



start
q->link=NULL;

1000 q 2000

start

1000 q 2000

NULL

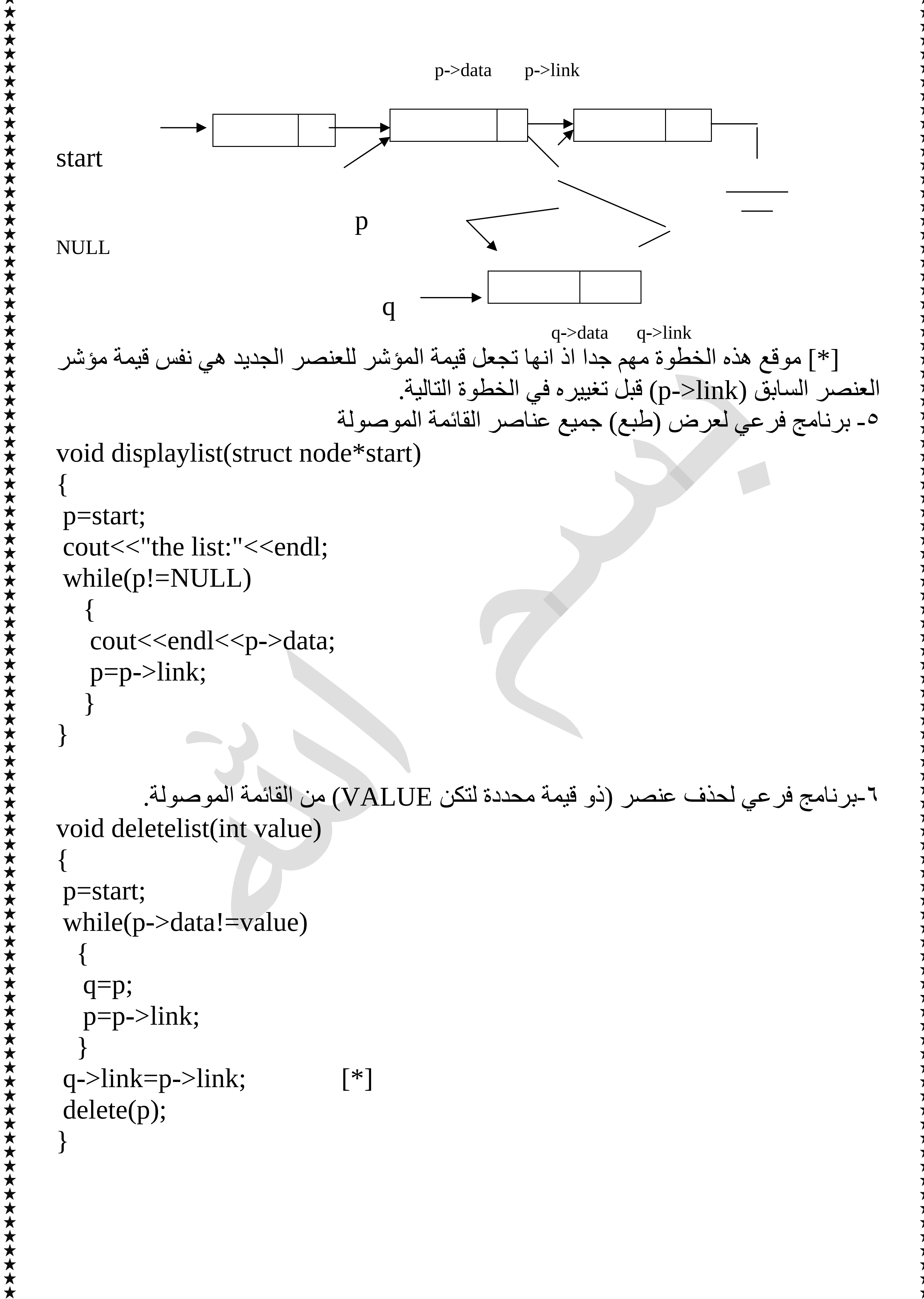
٣- تكوين قائمة موصولة تحتوي N من العناصر

```
void createNnode(struct node*start)
{
 int n,i;
 p=new node;
 start=p;
 cout<<"how many elements you like to create"<<endl;
 cin>>n;
 for(i=0;i<n;i++)
  {
   cin>p->data;
   if(i!=n-1)
   q=new node;
   else q=NULL;
   p->link=q;
   p=q;
  }
}
```

٤-برنامج فرعي لاضافة عنصر الى القائمة الموصولة بعد الموقع الذي يشير اليه المؤشر P

```
void insertafter(struct node*p)
{
 q=new node;
 cin>>q->data;
 q->link=p->link;            [*]
 p->link=q;
}
```

الرسم التوضيحي:



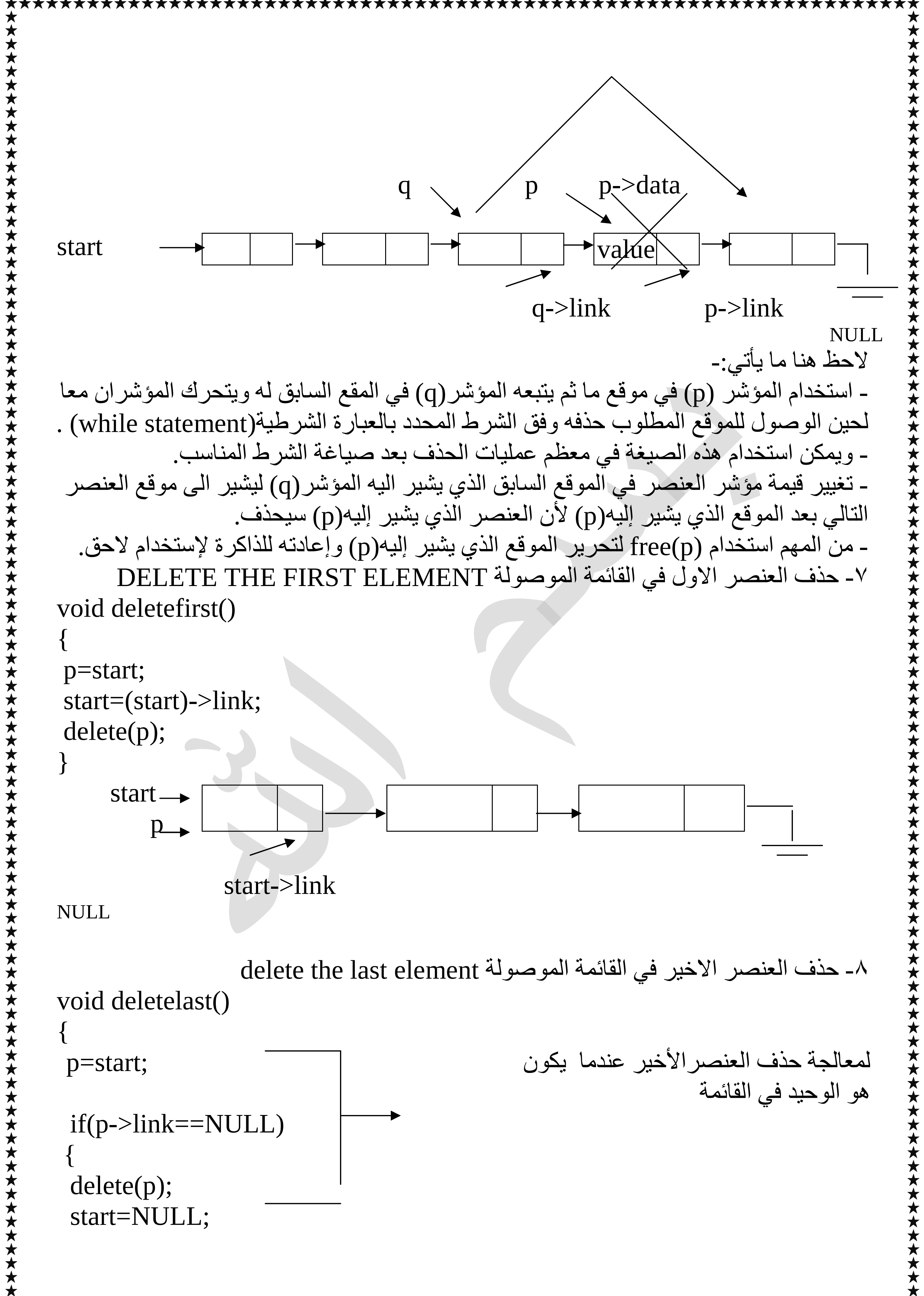
[*] موقع هذه الخطوة مهم جدا اذ انها تجعل قيمة المؤشر للعنصر الجديد هي نفس قيمة مؤشر العنصر السابق (p->link) قبل تغييره في الخطوة التالية.

٥- برنامج فرعي لعرض (طبع) جميع عناصر القائمة الموصولة

```
void displaylist(struct node*start)
{
 p=start;
 cout<<"the list:"<<endl;
 while(p!=NULL)
    {
     cout<<endl<<p->data;
     p=p->link;
    }
}
```

٦-برنامج فرعي لحذف عنصر (ذو قيمة محددة لتكن VALUE) من القائمة الموصولة.

```
void deletelist(int value)
{
 p=start;
 while(p->data!=value)
   {
    q=p;
    p=p->link;
   }
 q->link=p->link;              [*]
 delete(p);
}
```



لاحظ هنا ما يأتي:-

- استخدام المؤشر (p) في موقع ما ثم يتبعه المؤشر(q) في المقع السابق له ويتحرك المؤشران معا لحين الوصول للموقع المطلوب حذفه وفق الشرط المحدد بالعبارة الشرطية(while statement) .
- ويمكن استخدام هذه الصيغة في معظم عمليات الحذف بعد صياغة الشرط المناسب.
- تغيير قيمة مؤشر العنصر في الموقع السابق الذي يشير اليه المؤشر(q) ليشير الى موقع العنصر التالي بعد الموقع الذي يشير إليه(p) لأن العنصر الذي يشير إليه(p) سيحذف.
- من المهم استخدام free(p) لتحرير الموقع الذي يشير إليه(p) وإعادته للذاكرة لإستخدام لاحق.

٧- حذف العنصر الاول في القائمة الموصولة DELETE THE FIRST ELEMENT

```
void deletefirst()
{
 p=start;
 start=(start)->link;
 delete(p);
}
```

٨- حذف العنصر الاخير في القائمة الموصولة delete the last element

```
void deletelast()
{
  p=start;

  if(p->link==NULL)
  {
  delete(p);
  start=NULL;
```

لمعالجة حذف العنصرالأخير عندما يكون هو الوحيد في القائمة

```
 }
 else
 {
  while(p->link!=NULL)
     {
      q=p;
      p=p->link;
     }
  q->link=NULL;
  delete(p);
  }
}
```

start → [|] → [|] → [|] (q) → [|] (p) free

NULL

تمرين : اكتب برنامج فرعي procedure لقلب invert ترتيب عناصر القائمة الموصولة فمثلا اذا كان x يشير الى عناصر القائمة بحيث $(x=(a_1,a_2,a_3,\ldots a_n))$ فالمطلوب ان تصبح :

$X= (a_n, a_{n-1},\ldots.a_2,a_1)$

```
void invert(struct node **x)
{
 p=*x;
 q=NULL;
 while(p!=NULL)
    {
     r=q;
     q=p;
     p=p->link;
     q->link=r;
    }
 *x=q;
 cout<<"start="<<endl<<*start;
 displaylist(&start);
}
```

٩- اضافة عنصر واحد الى نهاية القائمة الموصولة

```
void addend()
{
 p=start;
 while(p->link!=NULL)
 p=p->link;
 cin>>q->data;
 q=new node;
 q->link=NULL;
 p->link=q;
}
```

١٠- اضافة عنصر بعد الموقع الذي ترتيبه n في القائمة الموصولة (عناصر القائمة اكبر او يساوي n) .

```
void addafterNelement()
{
 int n,i;
 cout<<endl<<"input the position(n)"<<endl;
 scanf("%d",&n);
 p=start;
 for(i=0;i<n-1;i++)
 p=p->link;
 q=new node;
 cin>>q->data;
 q->link=p->link;
 p->link=q;
}
```

١١-اضافة عنصر قبل العنصر في الموقع p للقائمة الموصولة.

ان الحالة تفترض ان المؤشر الرئيسي للقائمة مجهول وكذلك بيانات العناصر مجهولة لذا فان فكرة الحل هي:

+ انشاء العنصر الجديد
+ إضافة العنصر الجديد بعد العنصر في الموقع (p)
+ إستبدال قيمة العنصر الجديد مع قيمة العنصر في الموقع(p)

```
void addbefore(struct node*p)
{
 q=new node;
 q->data=p->data;
```

قراءة العنصر الجديد وليكن M ليحل بدلا من قيمته السابقة C

```
 cin>>p->data;
```

```
 q->link=p->link;
 p->link=q;
}
```

B C M D

P

NULL

C q

لاحظ ان عناصر القائمة أصبحت BMCD بعد ان كانت BCD

١٢- حذف عنصر في الموقع (p) من القائمة الموصولة

```
void deletenodep(struct node*p)
{
 q=p->link;
 p->data=q->data;
 p->link=q->link;
delete(q);
}
```

p q

NULL

تمرين : اضافة عنصر واحد الى قائمة موصولة عناصرها مرتبة ordered linked list بصورة تصاعدية ascending على ان تبقى عناصر القائمة مرتبة بعد الاضافة.

```
void insertinascending()
{
 struct node*t=new node;
 int value;
 cin>>value;
 p=start;
 while(p->data<value && p!=NULL)
    {
     q=p;
     p=p->link;
    }
```

```
t->data=value;
t->link=p;
q->link=t;
if(value<=start->data)
start=t;
cout<<"start="<<start<<endl;
}
```

تمرين: استبدال exchange قيمة عنصر في موقع معين i للقائمة الموصولة start مع قيمة العنصر في موقع اخر j في نفس القائمة ، على ان تكون I<j

```
void swap(int *x,int *y)
{
  int temp;
  temp=*x;
  *x=*y;
  *y=temp;
 }
void exchange()
{
 int n,i,j,x;
 cout<<"input the locations you want to exchange"<<endl;
 cin>>i>>j;
 p=start;
 for(n=1;n<=(i-1);n++)
 p=p->link;
 cout<<endl<<"p="<<p->data;
 q=p->link;
 for(n=i+1;n<=(j-1);n++)
 q=q->link;
 cout<<endl<<"q="<<q->data);
 swap(&p->data,&q->data);
}
```

تمرين: اكتب برنامج فرعي لتنفيذ عملية دمج merge القائمة الموصولة التي مؤشرها الرئيسي Y في نهاية عناصر القائمة الموصولة التي مؤشرها الرئيسي x

```
void merge(struct node*x,struct node*y)
{
 struct node*z;
 if(x==NULL)
```

```
z=y;
else
{
 z=x;
 if(y!=NULL)
  {
   p=x;
   while(p->link!=NULL)
   p=p->link;
   p->link=y;
  }
 }
}
```

تمرين : اكتب برنامج فرعي لتجزئة split قائمة موصولة مؤشرها الرئيسي start الى قائمتين موصولتين احدهما first تحتوي على جميع العناصر في المواقع الفردية للقائمة الاصلية ، والقائمة الثانية second تحتوي على جميع العناصر في الموقع الزوجية للقائمة الاصلية.

```
void split(struct node *start)
{
 struct node*m;
 struct node*n;
 int l;
 l=0;
 p=start;
 first=NULL;
 second=NULL;
 while(p!=NULL)
   {
    l++;
    if((l%2)!=0)
     {
      if(l==1)
        {
         first=p;
         n=first;
        }
        else
        {
```

```
          n->link=p;
          n=p;
         }
        }
        else
          {
           if(l==2)
           {
            second=p;
            m=second;
           }
          else
           {
            m->link=p;
            m=p;
            }
           }
         p=p->link;
    }
     n->link=NULL;
     m->link=NULL;
}
```

برنامج -٦ تمثيل القائمة الموصولة linked list وعدد من عمليات الاضافة والحذف

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node{
                int data;
                struct node *link;
               }*p,*q,*f,*t,*start;
void createNnode()
{
 int n,i;
 p=new node;
 start=p;
 cout<<"how many elements you like to create"<<endl;
 cin>>n;
```

```
 for(i=0;i<n;i++)
  {
        cin>>p->data;
        if(i!=n-1)
        q=new node;
        else q=NULL;
        p->link=q;
        p=q;
  }
}
void addafter(int item,struct node*q)
{
 p=new node;
 p->data=item;
 p->link=q->link;
 q->link=p;
}
void addbefore(int item,struct node*p)
{
 q=new node;
 q->data=p->data;
 p->data=item;
 q->link=p->link;
 p->link=q;
}
void deletelist(int value)
{
 p=start;
 while(p->data!=value)
        {
         q=p;
         p=p->link;
        }
 q->link=p->link;
 delete(p);
}
void deletenodep(struct node*p)
```

```
{
 q=p->link;
 p->data=q->data;
 p->link=q->link;
 delete(q);
}
void displaylist()
{
 p=start;
 while(p!=NULL)
         {
          cout<<p->data<<"\t";
          p=p->link;
         }
}
void main()
{
 int choice,k,item,item1,l;
 clrscr();
 start=NULL;
 do
  {
        cout<<"representation of linked list and its operation"<<endl;
        cout<<"---------------------------------------------"<<endl;
        cout<<"1-creation alinked list"<<endl;
        cout<<"2-insertion after a creation position(element)"<<endl;
        cout<<"3-insertion before a creation position(element)"<<endl;
        cout<<"4-deletion an element of creation value"<<endl;
        cout<<"5-deletion an element(s) at creation position"<<endl;
        cout<<"6-display the content of the linked list"<<endl;
        cout<<"7-exit"<<endl;
        cout<<"select your choice"<<endl;
        cin>>choice;
        switch(choice)
                {
                 case(1):
                 createNnode();
```

```
                break;
                case(2):
            {
             cout<<"give the element where to insert the new item after
it"<<endl;
             cin>>item;
             f=start;
             while(f->data!=item)
             f=f->link;
             cout<<"how many elements you like add"<<endl;
             cin>>k;
             for(l=0;l<k;l++)
                 {
                  cout<<"enter the new element"<<endl;
                  cin>>item1;
                  addafter(item1,f);
                  }
             break;
             }
             case(3):
                 {
                  cout<<"give the element where to insert the new before after
it"<<endl;
                  cin>>item;
                  f=start;
                  while(f->data!=item)
                  f=f->link;
                  cout<<"how many elements you like to enter"<<endl;
                  cin>>k;
                  for(l=0;l<k;l++)
                      {
                 cout<<"enter the new element"<<endl;
                 cin>>item1;
                 addbefore(item1,f);
                 }
                      break;
                      }
```

```
                case(4):
                        {
              cout<<"give the value of the element you like to delete"<<endl;
              cin>>item;
              deletelist(item);
              break;
              }
                     case(5):
              {
               cout<<"give the position(sequence)of the element you like to
delete"<<endl;
               cin>>k;
               t=start;
               for(l=0;l<k-1;l++)
                      {
                       f=t;
                       t=t->link;
                      }
                cout<<"how many elements you like to delete"<<endl;
                cin>>k;
                for(l=0;l<k;l++)
                deletenodep(t);
                break;
                }
              case(6):
                {
                     cout<<"the element of the liked list are:"<<endl;
                     displaylist();
                     break;
                     }
                }
  }while(choice!=7);
}
```

برنامج ٧- : تمثيل القائمة الموصولة linked list وعمليتي طباعة عناصرها بصورة معكوسة reverse وقلب invert ترتيب عناصرها بشكل معكوس.

```
#include<iostream.h>
#include<conio.h>
```

```
#include<stdlib.h>
struct node{
            int data;
            struct node *link;
           }*p,*q,*f,*r,*start;

void createNnode()
{
 int n,i;
 p=new node;
 start=p;
 cout<<"how many elements you like to create"<<endl;
 cin>>n;
 for(i=0;i<n;i++)
  {
  cin>>p->data;
  if(i!=n-1)
  q=new node;
  else q=NULL;
  p->link=q;
  p=q;
  }
}
void displaylist()
{
 p=start;
 while(p!=NULL)
    {
     cout<<"\t"<<p->data;
     p=p->link;
    }
}
void invert(struct node **x)
{
 p=*x;
 q=NULL;
 while(p!=NULL)
```

```
    {
     r=q;
     q=p;
     p=p->link;
     q->link=r;
    }
 *x=q;
}
void printrev(struct node*p)
{
 if(p!=NULL)
  {
  printrev(p->link);
  cout<<"\t"<<p->data;
  }
 }
void main()
{
 int choice,k,item,item1,l;
 clrscr();
 start=NULL;
 do
  {
   cout<<"representation of linked list and its operation"<<endl;
   cout<<"----------------------------------------------------"<<endl;
   cout<<"1-creation alinked list"<<endl;
   cout<<"2-display the content of the linked list"<<endl;
   cout<<"3-print the elements in reverse order"<<endl;
   cout<<"4-reverse the order of the list element"<<endl;
   cout<<"5-exit"<<endl;
   cout<<"select your choice"<<endl;
   cin>>choice;
   switch(choice)
      {
       case(1):
       createNnode();
       break;
```

```
        case(2):
          {
          cout<<"the element of the liked list are:"<<endl;
          displaylist();
          break;
          }
          case(3):
            {
            cout<<"this is the elements of the list are printed in reverse
order"<<endl;
            printrev(start);
            break;
            }
            case(4):
              {
                invert(&start);
                cout<<"the elements of the list are reversed"<<endl;
                break;
                }
          }
}while(choice!=5);
}
```

A

٤-٤ المكدس الموصول linked stack

يمكن الاستفادة من خصائص الخزن الديناميكي لتمثيل المكدس باعتباره حالة خاصة من القائمة الخطية التي تكون عمليات الاضافة والحذف من نهاية واحدة هي النهاية المفتوحة.

ان مبدا عمليات الحذف والاضافة هي نفسها التي سبق ذكرها الا ان الفرق يكون في طريقة التمثيل في الذاكرة .والشكل التالي يبين مكدس موصول ذو اربعة عناصر .

start

C

B

D

NULL

- المؤشر Start: يشير الى قيمة المكدس حيث العنصر d وهو يمثل النهاية المفتوحة حيث تنفذ عمليات الاضافة والحذف.

- العنصر a في قعر المكدس ويمثل النهاية المغلقة مع ملاحظة ان حقل المؤشر لهذا العنصر هو NULL اذ لم يسبقه شيء.

<u>البرنامج الفرعي لاضافة عنصر الى المكدس الموصول</u>

```
struct node{
        int data;
        struct node*link;
        }*start,*p,*q;
void push()
{
 p=new node;
 cout<<"input element"<<endl;
 cin>>p->data;
 if(start==NULL)
 p->link=NULL;
 else
 p->link=start;
 start=p;
}
```

إن هذا البرنامج الفرعي يعتمد التعريف الوارد في الصفحة() فيما يتعلق بعناصر المكدس مع ملاحظة ما يلي:-

١- إن المؤشر (start) هو المؤشر الرئيس لبداية عناصر المكدس أي المؤشر الذي يناظر المؤشر (top).

٢- لا نحتاج إلى خطوة للتحقق من إمتلاء المكدس(stack full) لأننا نستطيع خلق العنصر عند الحاجة إليه و من ثم ربطه بالعنصر السابق له.

٣- بموجب أول إيعازين سيخلق العنصر المطلوب إضافته (المؤشر p يشير إليه) و تدخل بياناته.

٤- عبارة ;p->link=NULL هي لمعالجة الحالة عند خلق أول عنصر في المكدس.

٥- عبارة else هي لجعل قيمة حقل المؤشر للعنصر الجديد (المطلوب اضافته) تشير الى موقع العنصر السابق له و الذي يشير إليه(start).

٦- الخطوة الأخيرة هي لتحديث المؤشر (start) ليشير إلى العنصر الجديدبعد أن أصبح في مقدمة المكدس.

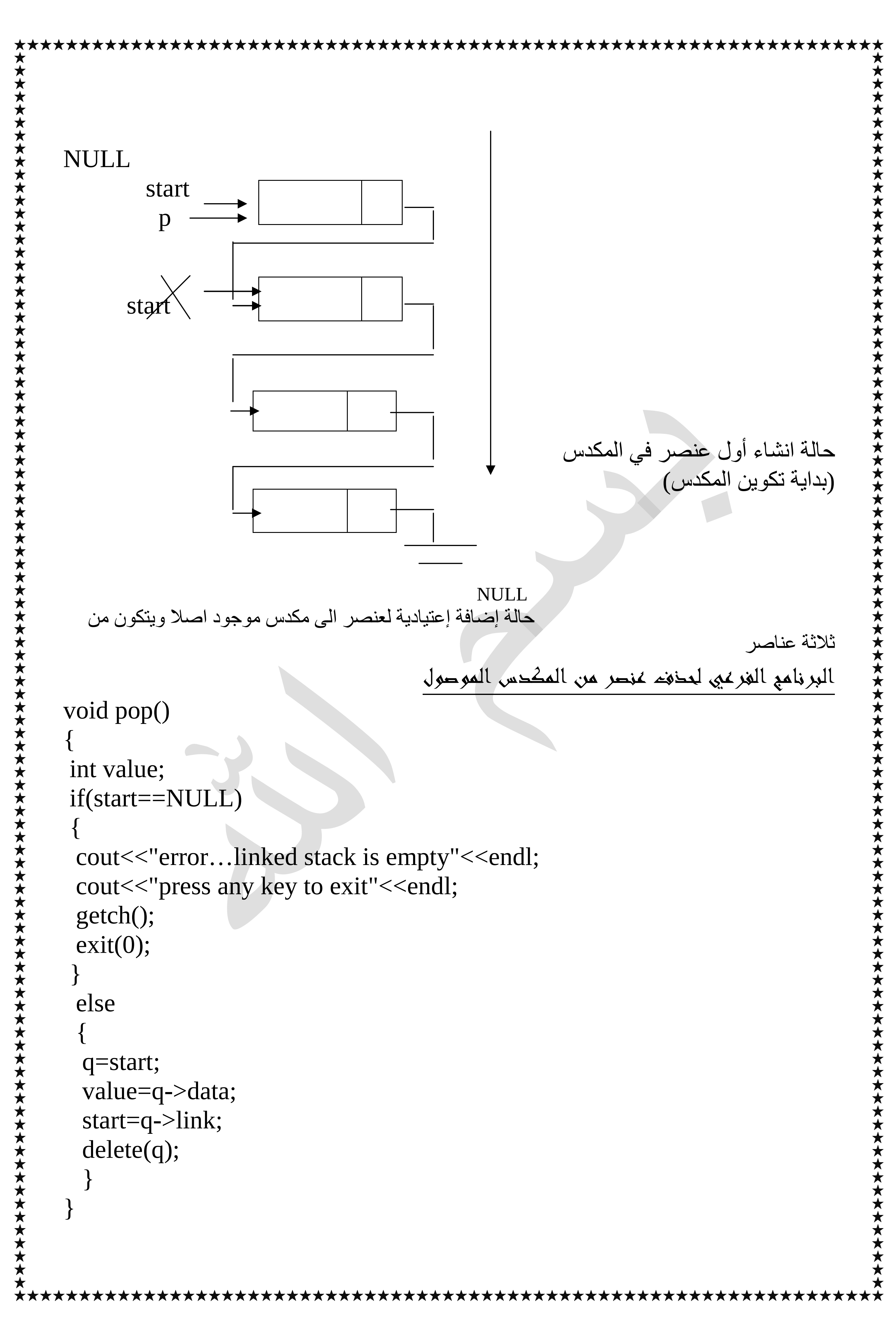
٧- الرسم التوضيحي يبين كيفية تنفيذ الخطوات أعلاه..

p->data p->link

start

p

p->data p->link



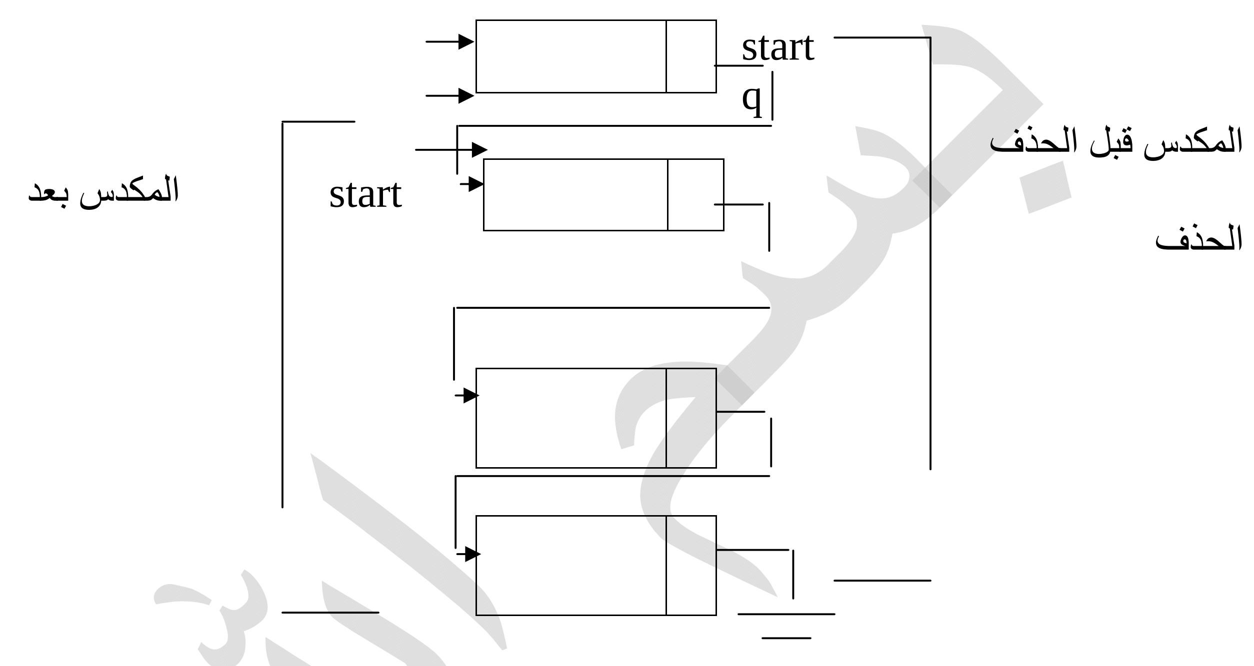
حالة انشاء أول عنصر في المكدس
(بداية تكوين المكدس)

حالة إضافة إعتيادية لعنصر الى مكدس موجود اصلا ويتكون من ثلاثة عناصر

<u>البرنامج الفرعي لحذف عنصر من المكدس الموصول</u>

```
void pop()
{
 int value;
 if(start==NULL)
  {
   cout<<"error…linked stack is empty"<<endl;
   cout<<"press any key to exit"<<endl;
   getch();
   exit(0);
  }
   else
   {
    q=start;
    value=q->data;
    start=q->link;
    delete(q);
    }
}
```

ان خطوات هذا البرنامج الفرعي هي:
١- عندما يكون المؤشر الرئيسي (start=NULL) فان المكدس خال وعملية الحذف غير ممكنة.
٢- استخدام المؤشر (q) ليشير الى بداية المكدس(اول عنصر في المكدس).
٣- أخذ(سحب) قيمة العنصر الأول الموجودة في الحقل(q->data) وخزنها وقتيا في المتغير (value).
٤- تحديث قيمة المؤشر (start) ليشير الى موقع العنصر التالي المحددة بالحقل (q->link).
٥- تحرير الموقع الذي كان يشغله العنصر المحذوف والذي يشير إليه المؤشر(q) باستخدام (free(q)).
٦- الرسم التوضيحي يبين كيفية تنفيذ الخطوات أعلاه:



برنامج ٨- تمثيل المكدس الموصول linked stack وعملياته

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node{
            int data;
            struct node*link;
            }*start,*p,*q;
void push( )
{
  p=new node;
  printf("\ninput element\n");
  cin>>p->data;
```

```
   if(start==NULL)
   p->link=NULL;
   else
   p->link=start;
   start=p;
 }
 void pop()
 {
  int value;
  if(start==NULL)
  {
   cout<<"error...linked stack is empty"<<endl;
   cout<<"press any key to exit"<<endl;
   getch();
   exit(0);
  }
   else
   {
    q=start;
    value=q->data;
    start=q->link;
    delete(q);
    }
 }
 void main()
 {
  int choice,l,m,i,item1;
  clrscr( );
  start=NULL;
  do
   {
    cout<<"representation of linked stack operation"<<endl;
    cout<<"--------------------------------"<<endl;
    cout<<"1-push a new element(s)      "<<endl;
    cout<<"2-pop an element        "<<endl;
    cout<<"3-display the content of the linked stack"<<endl;
    cout<<"4-exit                              "<<endl;
```

```
    cout<<"select your choice"<<endl;
    cin>>choice;
    switch (choice){
                  case(1):
                    {
                     cout<<"how many elements you like to enter";
                     cin>>m;
                     for(i=0;i<m;i++)
                       {
                        push();
                       }
                        break;
                     }
                   case(2):
                    {
                     cout<<endl<<"how many elements you want to
  delete"<<endl;
                     cin>>l;
                     for(i=0;i<l;i++)
                     pop( );
                     break;
                     }
                     case(3):
                       {
                        if(start==NULL)
                        cout<<"error...linked stack is empty"<<endl;
                        else
                         {
                          cout<<"the content of the linked  stack is:"<<endl;
                          q=start;
                          while(q!=NULL)
                            {
                             cout<<endl<<q->data<<endl;
                             q=q->link;
                             }
                         }
                        break;
```

```
                    }
              }
 }while(choice!=4);
}
```

٤-٥ الطابور الموصول linked queue

كما سبق ان مثلنا المكدس باستخدام الخزن الديناميكي يمكن تمثيل الطابور بنفس الطريقة مع وجود المؤشرين rear , front ويظهر الطابور كالاتي:

rear — front
D ← C ← B ← A
NULL

- المؤشر front يشير الى اول عنصر في الطابور.
- المؤشر rear يشير الى اخر عنصر في الطابور.
- كل عنصر من عناصر الطابور الاربعة a,b,c,d فيه حقل يحتوي قيمة المؤشر الى العنصر التالي.
- مؤشر العنصر الاخير قيمته nil اذ لا يوجد بعده عناصر

البرنامج الفرعي لاضافة عنصر الى الطابور الموصول:

```
struct node{
            int data;
            struct node*link;
           }*rear,*front,*p,*q;
void add()
{
 p=new node;
 cout<<"input new element"<<endl;
 cin>>p->data;
 p->link=NULL;
 if(rear==NULL)
 front=p;
 else rear->link=p;
 rear=p;
}
```

ان خطوات هذا البرنامج الفرعي هي كالآتي:

١- ان الخطوات الثلاث الاولى تمثل خلق العنصر وادخال قيمته وجعل المؤشر (NULL).

٢- عبارة (front=p) هي لمعالجةالحالة عند خلق اول عنصر في الطابور وسيشير إليه المؤشر (front).

٣- عبارة (else) هي تحديث قيمة حقل المؤشر للعنصر الأخير في الطابور وجعله يشير الى العنصر الجديد الذي يشير اليه (p).

٤- الخطوة الأخيرة تحديث قيمة حقل المؤشر (rear) ليشير الى العنصر الجديد(المضاف) بعد ان اصبح هو الأخير.

٥- الرسم التوضيحي التالي لإضافة العنصر D.

rear p rear front

D C B A

NULL NULL

٦- الرسم التوضيحي التالي يبين تكوين أول عنصر في الطابور هو (A).

P rear front

A

p->link p->data

NULL

<u>البرنامج الفرعي لحذف عنصر من الطابور الموصول:</u>

```
void deleteq()
{
 int item;
 p=front;
 if(p==NULL)
 {
  cout<<"error...the linked queue is empty"<<endl;
  cout<<"press any key to exit"<<endl;
  getch();
  exit(0);
 }
 else
 {
  q=p->link;
  item=p->data;
  free(p);
  front=q;
  if(front==NULL)
```

```
 rear=NULL;
 }
}
```

ان خطوات هذا البرنامج الفرعي هي كالآتي:

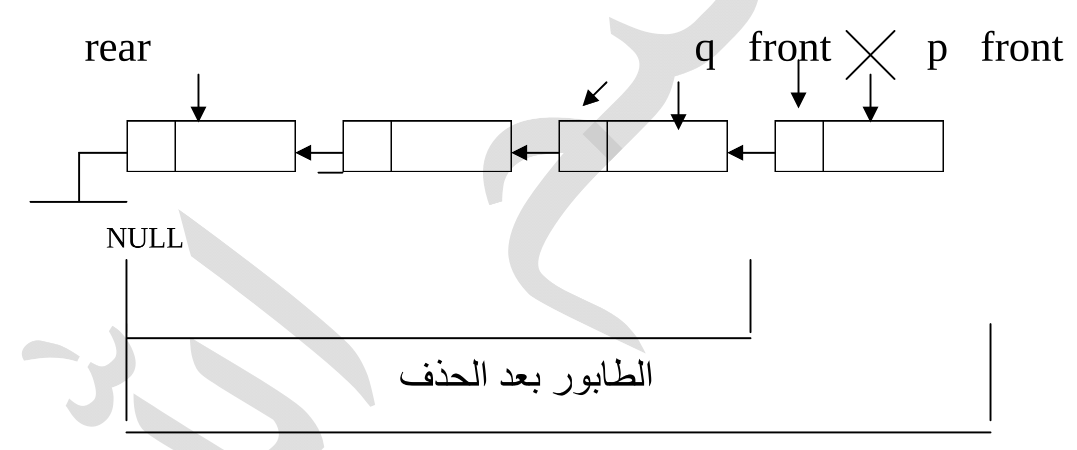
١- استخدام مؤشر وقتي (p) ليشير إلى أول عنصر في الطابور حيث يشير المؤشر (front) وعندما تكون قيمته (NULL) فهذا يعني أن الطابور خالي من العناصر ولا يمكن تنفيذ عملية الحذف.

٢- في مقدمة عبارة (else) نستخدم مؤشر ثان هو (q) يشير إلى العنصر الثاني في الطابور لكي نستطيع حذف العنصر الأول بعد خزن قيمته وقتيا في المتغير (item).

٣- الخطوة الرابعة في عبارة (else) هي لتحديث قيمة المؤشر (front) ليشير إلى موقع العنصر الثاني حيث يشير (q).

٤- الخطوتان الأخيرتان هي لمعالجة حذف آخر عنصر في الطابور مما يتطلب جعل قيمة كل من المؤشرين (front)،(rear) هي (NULL).

٥- الرسم التوضيحي التالي يبين كيفية تنفيذ الخطوات أعلاه:



الطابور قبل الحذف

تمرين : اكتب برنامج فرعي لنسخ جميع عناصر المكدس المتسلسل sequential stack الى طابور موصول linked queue خال من العناصر بحيث اعلى عنصر في المكدس يصبح اول عنصر في الطابور.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
const size=30;
struct node{
        int data;
        struct node*link;
        }*front,*rear,*p,*start;
int t,top;
int st[size];
```

```
void copy1()
{
 int item;
 t=top;
 while(t!=-1)
    {
     item=st[t];
     t--;
     p=new node;
     p->data=item;
     p->link=NULL;
     if(rear==NULL)
     front=p;
     else
     rear->link=p;
     rear=p;
      }
}
```

تمرين : اكتب برنامج فرعي لنسخ جميع عناصر الطابور الموصول linked queue الى مكدس متسلسل sequential stack خال من العناصر بحيث اول عنصر في الطابور يصبح اعلى عنصر في المكدس.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
const size=10;
struct node{
            int data;
            struct node*link;
            }*start,*rear,*front,*p;
int st1[size],st2[size];
int t=-1;
int top=-1;
void copy2()
{
 start=front;
 while(start!=NULL)
   {
```

```
   p=start;
   t++;
   st1[t]=p->data;
   start=start->link;
   }
 while(t!=-1)
  {
   top++;
   st2[top]=st1[t];
   t--;
   }
}
```

<u>برنامج ٩- تمثيل الطابور الموصول linked queue وعملياته</u>

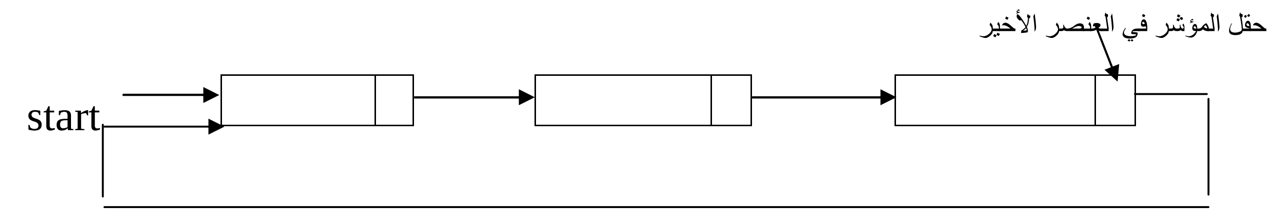
```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node{
            int data;
            struct node*link;
            }*rear,*front,*p,*q;
void add()
{
 p=new node;
 cout<<"input new element"<<endl;
 cin>>p->data;
 p->link=NULL;
 if(rear==NULL)
 front=p;
 else rear->link=p;
 rear=p;
}
void deleteq()
{
 int item;
 p=front;
```

```
if(p==NULL)
{
 cout<<"error...the linked queue is empty"<<endl;
 cout<<"press any key to exit"<<endl;
 getch();
 exit(0);
}
else
{
 q=p->link;
 item=p->data;
 delete(p);
 front=q;
 if(front==NULL)
 rear=NULL;
}
}
void main()
    {
        int choice,l,m,i;
        front=NULL; rear=NULL;
        clrscr();
        do{
          cout<<"representation of the linked queue operations"<<endl;
          cout<<"--------------------------------"<<endl;
          cout<<"1-add a new element     "<<endl;
          cout<<"2-delete an element       "<<endl;
          cout<<"3-display the content of the linked queue"<<endl;
          cout<<"4-exit                               "<<endl;
           cout<<"select your choice"<<endl;
          cin>>choice;
          switch (choice)
          {
                    case(1):
               {
                         cout<<"how many elements you like to enter";
                         cin>>m;
```

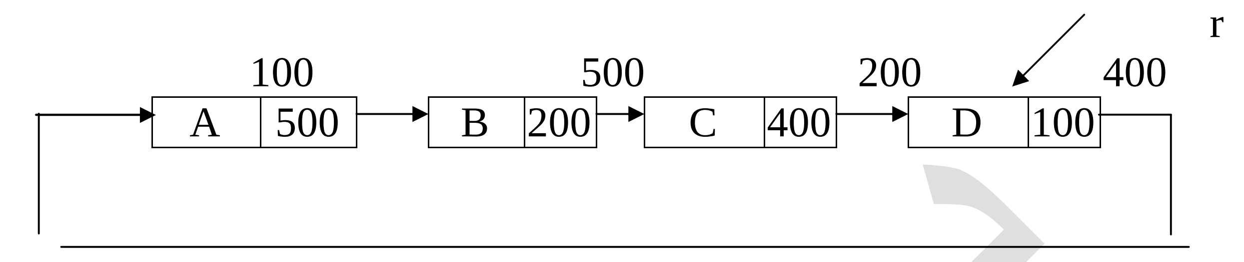
```
                        for(i=0;i<m;i++)
                        add( );
                        break;
                       }
                 case(2):
                {
                        cout<<endl<<"how many elements you want to delete";
                        cin>>l;
                        for(i=0;i<l;i++)
                        deleteq();
                        break;
                       }
                case(3):
                {
                        if(front==NULL)
                        cout<<"error...the linked queue is empty"<<endl;
                        else
                         {
                           q=front;
                         cout<<"the content of the stack is:"<<endl;
                         while(q!=NULL)
                            {
                             printf("\n%d\n",q->data);
                             q=q->link;
                            }
                           break;
                          }
                        }
             }
          }while(choice!=4);
      }
```

٤-٦ القائمة الموصولة الدائرية (الهيكل الدائري) circular linked list (ring structure)

في القائمة الموصولة الاعتيادية نستخدم مؤشر رئيسي يشير الى موقع اول عنصر ، وحقل المؤشر في العنصر الاخير تكون قيمته nil اذ لا يتبعه عنصر اخر. في هذا الهيكل الدائري circular linked listفان حقل المؤشر في العنصر الاخير سيشير الى العنصر الاول في القائمة كما في الشكل التالي:



في هذه الحالة يمكن استخدام مؤشر واحد فقط يشير إلى العنصر الأخير وليكن r وبدلالته نستطيع الوصول إلى العنصر الأول كما في الشكل التالي:



ان المؤشر r ستكون قيمته 400 ليشير إلى العنصر الأخير ، وحقل المؤشر للعنصر الأخير هو (r->link) نجد أن قيمته هي 100 وهذا يمثل عنوان موقع العنصر الأول.

<u>البرنامج الفرعي لاضافة عنصر الى القائمة الموصولة الدائرية</u>

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node{
            int data;
            struct node*link;
            }*start,*p,*r,*f,*q;
void addcll()
{
 p=new node;
 cout<<"input new element"<<endl;
 cin>>p->data;
 if(r==NULL)
 p->link=p;
 else
   {
    p->link=r->link;
    r->link=p;
    }
 r=p;
}
```

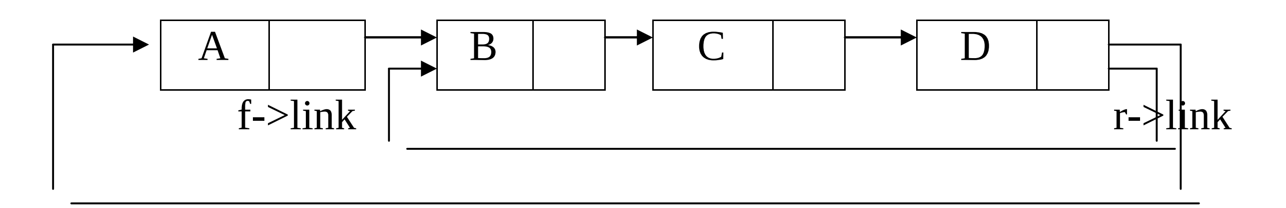
في حالة إضافة أول عنصر(تكوين القائمة لاول مرة) فالرسم التالي يوضح ذلك:

أما في حالة إضافة عنصر مثل E إلى قائمة موجودة أصلا فالشكل يكون كالآتي:

البرنامج الفرعي لحذف عنصر من القائمة الموصولة الدائرية

```
void deletcll()
{
 int item;
 if(r==NULL)
 cout<<"error..theC.L.L is empty"<<endl;
 else
   {
    f=r->link;
    item=f->data;
    if(r==f)
    r=NULL;
    else
    r->link=f->link;
    delete(f);
    }
}
```

والرسم التوضيحي لحالة حذف العنصر A :



r->link=f->link

برنامج فرعي لعرض (طبع) محتويات القائمة الموصولة الدائرية

```
void displaycll()
{
 p=r;
 if(p==NULL)
 cout<<"the C.L.L is empty"<<endl;
 else
 do
  {
   cout<<"\t"<<((p->link)->data);
   p=p->link;
  }while(p!=r);
}
```

لاحظ الصيغة المركبة لإيعاز الكتابة الثاني إذ أن (p->link) تعني موقع العنصر الأول أما العبارة بأكملها فتعني كتابة حقل البيانات في موقع العنصر الأول وهكذا بالتتابع لبقية العناصر بعد تحريك المؤشر (p).

٤-٧ القائمة الموصولة المزدوجة double linked list

في القائمة الموصولة الاعتيادية linked list هناك صعوبة في حذف العنصر الذي يشير اليه المؤشر p لانه يتعذر العودة الى العنصر السابق له لتغيير حقل المؤشر فيه ليشير الى العنصر اللاحق ، اي ان التحرك في هذه القائمة باتجاه واحد.

اما القائمة الموصولة المزدوجة double linked list فان كل عنصر فيها يحتوي على مؤشرين احدهما يشير الى موقع العنصر اللاحق والاخر يشير الى موقع العنصر السابق اي ان كل عنصر في القائمة يتكون من ثلاثة اجزاء ويعرف في لغة C كالآتي :

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node{
        int data;
```

```
struct node*Llink;
struct node*Rlink;
}*start,*p,*q;
```

فالقائمة التالية تتكون من ثلاثة عناصر هي C,B,A

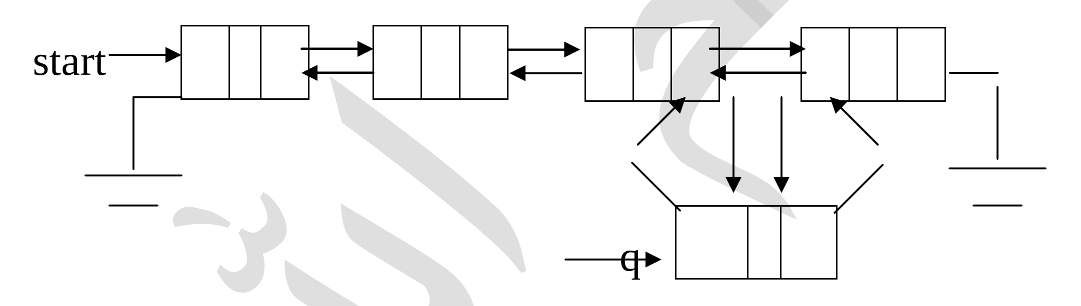
start → | Llink | A | Rlink | ⇄ | Llink | B | Rlink | ⇄ | Llink | C | Rlink |

NULL

NULL

<u>عملية اضافة عنصر بعد الموقع p</u>

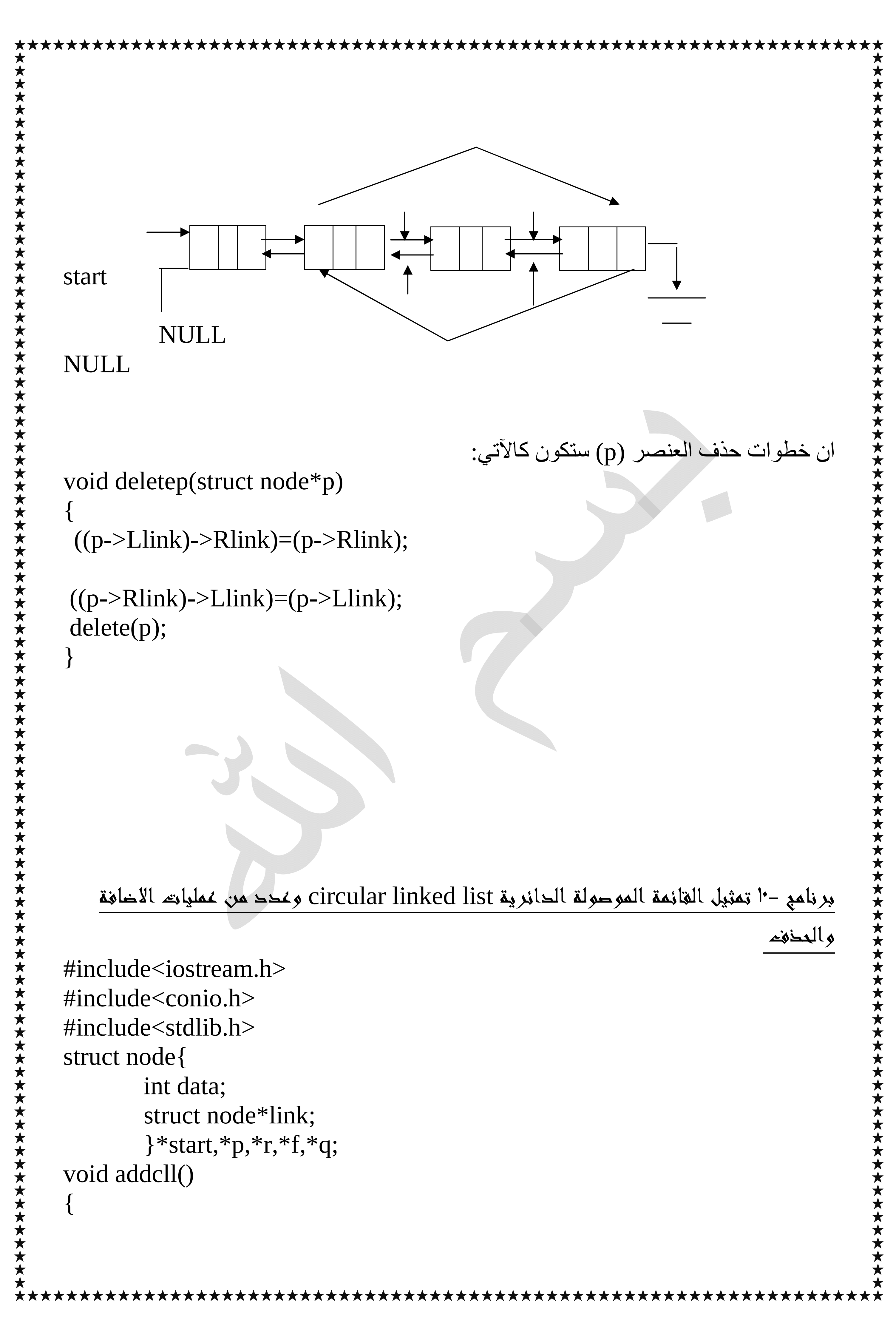
لنأخذ القائمة الموصولة المزدوجة التالية:-



فان سلسلة الايعازات في البرنامج الفرعي الآتي تمثل خطوات تنفيذ إضافة العنصر (q) إلى القائمة.

```
void addafter()
{
        p=new node;             تكوين العنصر :
 cin>>q->data ;                 قراءة البيانات
 q->Llink=p;                    ربط العنصر إلى يساره p:
  q->Rlink=p->Rlink;            ربط العنصر إلى يمين p:
 (p->Rlink)->Llink=q;           ربط العنصر اللاحق إلى العنصر الجديد:
 p->Rlink=q;                    ربط العنصر p إلى العنصر الجديد:
}
```

عملية حذف العنصر في الموقع p



ان خطوات حذف العنصر (p) ستكون كالآتي:

```
void deletep(struct node*p)
{
 ((p->Llink)->Rlink)=(p->Rlink);

 ((p->Rlink)->Llink)=(p->Llink);
 delete(p);
}
```

برنامج -١٠ تمثيل القائمة الموصولة الدائرية circular linked list وعدد من عمليات الاضافة والحذف

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node*link;
        }*start,*p,*r,*f,*q;
void addcll()
{
```

```
 p=new node;
 cout<<"input new element"<<endl;
 cin>>p->data;
 if(r==NULL)
 p->link=p;
 else
   {
    p->link=r->link;
   r->link=p;
    }
 r=p;
}
void deletcll()
{
 int item;
 if(r==NULL)
 cout<<"error..theC.L.L is empty"<<endl;
 else
   {
    f=r->link;    item=f->data;
    if(r==f)
    r=NULL;
    else
    r->link=f->link;
    delete(f);
    }
}
void displaycll()
{
 p=r;
 if(p==NULL)
 cout<<"the C.L.L is empty"<<endl;
 else
 do
  {
   Cout<<"\t"<<((p->link)->data);
   p=p->link;
```

```
  }while(p!=r);
 }
 void main()
 {
  int choice,l,m,i;
  r=NULL;
  clrscr();
 do
  {
   cout<<"representation of the C.L.L operations"<<endl;
   cout<<"-----------------------"<<endl;
   cout<<"1-add a new element(s)to the C.L.L "<<endl;
   cout<<"2-delete an element(s) from C.L.L"<<endl;
   cout<<"3-display the content of the  C.L.L"<<endl;
   cout<<"4-exit                          "<<endl;
   cout<<"select your choice"<<endl;
   cin>>choice;
   switch (choice)
       {
          case(1):
          {
                cout<<"how many elements you like to enter";
                cin>>m;
                for(i=0;i<m;i++)
                addcll();
             break;
                }
         case(2):
          {
                cout<<endl<<"how many elements you want to delete"<<endl;
                cin>>l;
                for(i=0;i<l;i++)
                deletcll();
                break;
             }
         case(3):
          {
```

```
            Cout<<"the content of C.L.L:"<<endl;
            displaycll();
          break;
            }
        }
  }while(choice!=4) ;
}
```

اسئلة الفصل

1- What do we mean by dynamic allocation of storage? Explain that in detail .what are its advantages and disadvantages?
2- Compare between sequential and dynamic allocation of storage.
3- What do we mean by a linked list?
4- Write a segment of code of c program for each of the following:
a- Create a linked list of two nodes.
b- Delete the first element in the linked list.
c- Insert one element after position p in a linked list.
d- Insert one element before position p in a linked list.
e- Delete all the elements where its data field is odd in any linked list.
f- Delete the last element of any linked list.
g- Delete all the elements of even position in the linked list.
h- Delete any zero value elements in the linked list.
i- Count the number of elements in the linked list.
5- given the following linked list:
Write a segment of code of c program to perform the followings:
a- Delete all the even numbers.
b- Add the element 70, 80, 90,100 before the last element in the list.
6- let a=(a1,a2….a10)and b=(b1,b2,….b8)Are two linked lists. Write a segment of code of c program to merge the first five elements of the list b at the end of the list a
7- write a segment of cod of c program to display all elements of any linked list except the element of data value44
8- write a segment of code of c program to delete all elements of the linked list except the first and last elements

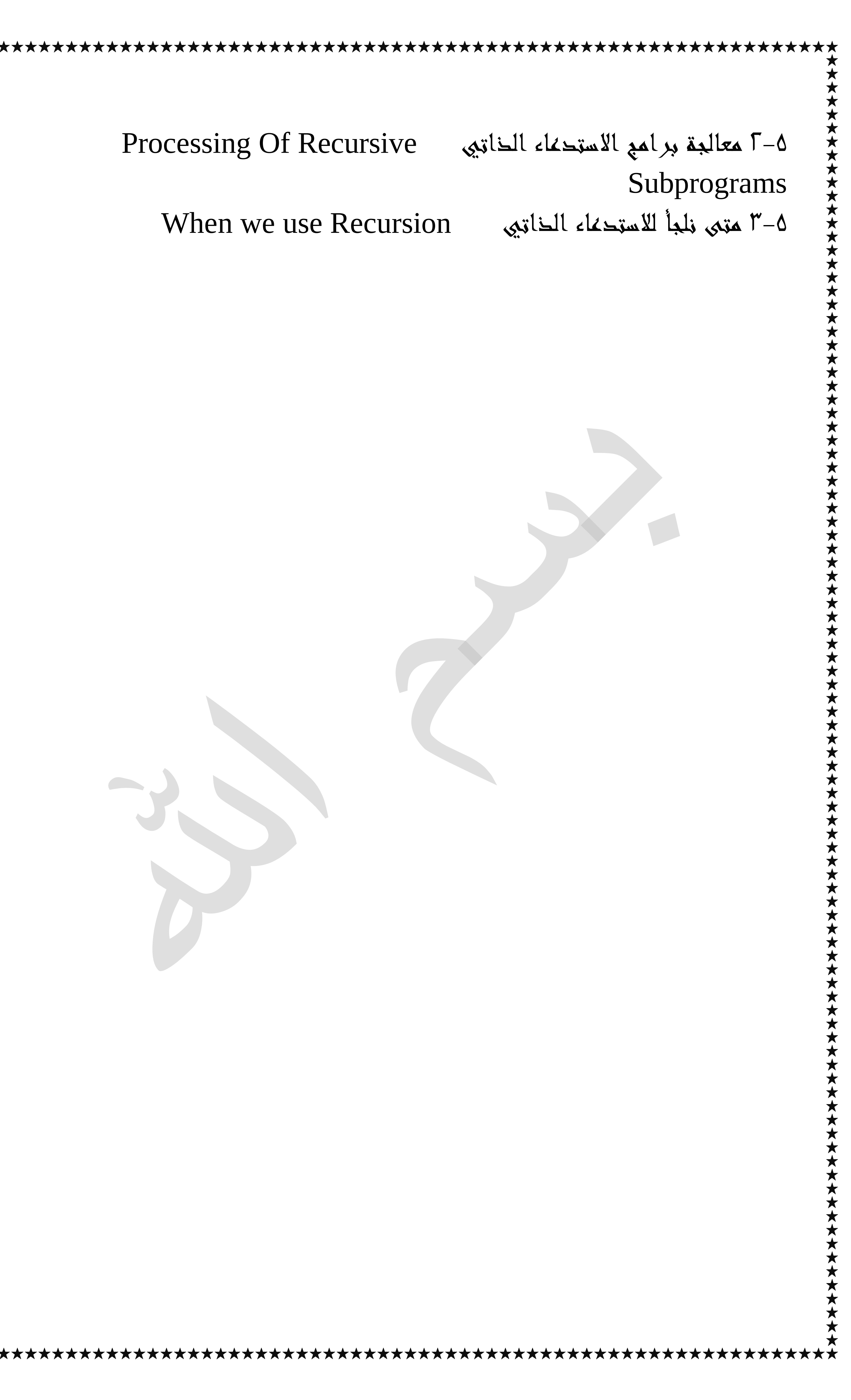
9- give the following linked list:
Write a segment of code of c program to:
a- increment the value of the fourth an fifth element by 50.
b- Add an element of value 60 before the last element in the list.
10- Give the output of the following code segment using the given linked list.
11- Given the following list
Write a segment of code of c program to:
a- Delete one element from it.
b- Delete two elements from it.
c- Add one element of data value40.
d- Display all the elements.
12- write a segment of code of c program too split any given linked list such that the first will consist of all elements with data field< value and the second will consist of all elements with data field>=value.
13- Show what the output of the following segment of code is:
14- Write a procedure to display all the elements of the linked list
15- Write a segment of code of c program to:
a- merge any two linked lists(x, y)
b- Merge any three linked lists (a, b, c)
16-write a procedure to find the largest element in the linked list pointed by r
17- Write a procedure to invert any given linked list
Note : assume the linked list is pointed by x and its elements A1,a2 ,…..an should be inverted to become X= an, an-1, …….., a2, a1
18- Write a segment of code of c program to :
a- delete an element at nth position from a linked list pointed by r.
b- Delete the element precedes the last element of the linked list pointed by r.
c- Delete the element after the node has information hh from the linked list pointed by r.
d- Insert one node (element) before the last node in a linked list pointed by r.
19- Write a recursive procedure to printout the element of the linked list in reverse order.
20- Given the following linked list.

Write a segment of code of c program for each of the following to be perform on the original list.
a- insert the element 66 before the element 6
b- delete the element 8
c- insert the element 77 after position p
d- delete the element 3
21-give the declaration required and writes a segment of code of c program to create a linked list of n student's names.
22- let Abe a linked list consist of 15 positive integers, write a segment of code of c program to spilt it in two lists, the first one consist of all the odd numbers and second one consist of all the even numbers.
23- Given a linked list consist of three elements A, B and C; let F is a pointer to the list head. What is the effect of each of the following segment of code?
24- Write a procedure to count the number of odd value elements in any linked list.
25- Write a function (search) to check the existence of a certain value in the linked list.

الفصل الخامس

Recursion الاستدعاء الذاتي

٥-١ تعريف الاستدعاء الذاتي Definition



٥-٢ معالجة برامج الاستدعاء الذاتي Processing Of Recursive Subprograms

٥-٣ متى نلجأ للاستدعاء الذاتي When we use Recursion

٥-١ تعريف الاستدعاء الذاتي

هو قابلية البرنامج الفرعي (function or procedure) لاستدعاء نفسه،انه مفهوم رياضي وطريقة برمجة قيمة،واسلوب برمجي فعال إذ يمكن استخدامه بدلا من استعمال التكرار(iteration).هنالك العديد من الصيغ الرياضية يمكن التعبير عنها باستخدام الاستدعاء الذاتي.

مثال ١: لاحتساب دالة مضروب العدد(n!)   [factorial of n] ، فانه يعرف رياضيا كالآتي:

$$n! = \begin{cases} 1 & \text{if } (n==0) \\ n*(n-1)! & \text{if}(n>0) \end{cases}$$

نجد ان (n!) تعرف بواسطة!(n-1) اي تعرف نفسها بصورة متكررة ذاتيا وتعني :
(مضروب العدد=العدد*مضروب العدد الذي يسبقه) . اي ان ايجاد مضروب العدد n يتطلب تكرار الدالة(n) من المرات باعتماد تعريف الدالة نفسه في كل مرة،كما في حساب مضروب العدد 5!

5!=5*(5-1)!
=5*4!
=5*4*(4-1)!
=5*4*3!
=5*4*3*(3-1)!
=5*4*3*2!
=5*4*3*2*(2-1)!
=5*4*3*2*1!
=5*4*3*2*(1-1)!
=5*4*3*2*1*0!
=5*4*3*2*1*1

ملاحظة:(0!=1) حسب التعريف الرياضي.

مثال ٢: لاحتساب دالة(X^m) power ،اي لاحتساب قيمة العدد (X) مرفوعا الى القوة(m).وهذه الدالة يمكن تعريفها باستخدام صيغة التكرار الذاتي:

$$f(X^m) = \begin{cases} 1 & \text{if}(m=0) \\ X*f(X^{m-1}) & \text{if}(m>0) \end{cases}$$

ولو اخذنا المثال العددي(2^4) مثلا:

2^4=2*2^(4-1)
=2*2^3
=2*2*2^(3-1)
=2*2*2^2
=2*2*2*2^(2-1)

=2*2*2*2^1
=2*2*2*2*2^(1-1)
=2*2*2*2*2^0

مثل هذه العمليات يمكن برمجتها باستخدام ما يعرف بالاستدعاء الذاتي(Recursion) كما يمكن برمجتها بالصيغة الأعتيادية دون استخدام الاستدعاء الذاتي فلو اخذنا المثال الأول عن دالة مضروب العدد (n!) فاننا يمكن أن نكتبها برمجيا بصيغة التكرار(iteration) وصيغة الاستدعاء الذاتي (Recursion).

أ- تعريف الدالة بدون استخدام الاستدعاء الذاتي

```
int fact(int n)
{
 int i,prod,fact;
 prod=1;
 for(i=1;i<=n;i++)
 prod*=i;
 fact=prod;
 return fact;
}
```

ب- تعريف الدالة باستخدام الاستدعاء الذاتي :

```
int fact1(int n)
{
 int fact;
 if(n<=1)
 fact=1;
 else
 fact=n*fact1(n-1);
}
```

ان استدعاء الدالة fact(n) ستنفذ على العدد (n) ، وعند تنفيذ جملة (else) فان الدالة ستستدعي نفسها على العدد (n-1) ، ومرة أخرى يستمر تنفيذ جملة (else)وتستدعي الدالة نفسها على العدد (n-2) وسيتوقف تنفيذ جملة (else) حين يصل الاستدعاء الى (١) fact اذ تحتسب النتيجة تراكميا وبشكل تراجعي ابتداءاً من هذه الخطوة ثم الخطوات التي تسبقها ولغاية الحصول على النتيجة النهائية . فلو نفذنا هذه الدالة على العدد (n=4) فان جملة else ستنفذ بصيغة تكرارية كما يأتي :

```
fact =4*fact (4-1)
      =4*3*fact (3-1)
```

=4*3*2*fact(2-1)

(2-1) !=1!=1

2

6

24 النتيجة النهائية

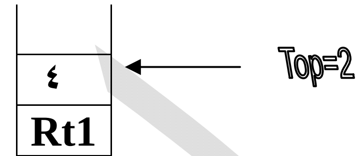
٢-٥ معالجة برامج الاستدعاء الذاتي Processing of Recursive Subprograms

لمعرفة متى يكون الاستدعاء الذاتي مفيدا علينا ان نفهم أولا كيفية معالجة لغات البرمجة للبرامج التي تحتوي برامج فرعية بصيغة الاستدعاء الذاتي . اذ يستخدم لهذا الغرض المكدس (stack) وفق الآتي :

فعند كل استدعاء للبرامج الفرعي(recursive procedure) أو (recursive function) يتم خزن (push) عنوان الرجوع (return address)في المكدس مع نسخ قيم المعالم (parameters)لذلك الاستدعاء ويتكرر هذا عند كل استدعاء للبرنامج الفرعي لحين الوصول الى حالة (base case)حيث تبدا العملية المعكوسة وهي اخراج (pop)محتويات المكدس بالتتابع والوصول الى النتيجة النهائية .

لناخذ البرنامج الفرعي fact (n) ونتابع حالة المكدس عند نتفيذه لاحتساب مضروب العدد (٤)اي [fact (4)]

١- عند أول استدعاء للبرنامج الفرعيfact(4) وسيخزن في المكدس كل من عنوان الرجوع(Rt1) والعدد (٤) .



٢- في جملة (else) تستدعي الدالة على العدد التالي (n-1) أي fact(٣) وعليه يخزن في المكدس عنوان الرجوع لهذا الاستدعاء (Rt2) والعدد (٣) .

٣

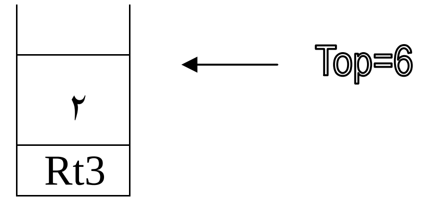
Rt2

٤

Rt1

Top=4

3_ وفي الاستدعاء اللاحق للعدد التالي n-1 أي fact(2) سيخزن في المكدس عنوان الرجوع له(Rt3) والعدد(2)



3
Rt2
٤
Rt1

٤- الاستدعاء اللاحق يكون للعدد التاليn-1 أيfact(1) وسيخزن في المكدس عنوان الرجوع له (Rt4)والعدد (1)

١
Rt4
٢
Rt3
3
Rt2
٤
Rt1

Top=8

٥-وعند تنفيذ البرنامج الفرعي على العدد (١) تنفذ جملة (fact=1) لان (n<2) لتعطي النتيجة (fact=1) وهي حالة توقف التكرار (Base Case) .

٦- يتم اخراج (POP)قيمة المتغير(n=1) واخراج عنوان الرجوع (Rt4)ثم ادخال نتيجة الاحتساب وهي (1).

٧-اخراج (POP)النتيجة الأخيرة وهي (١) وكذلك المتغير (n=2) مع عنوان الرجوع (Rt3) لاحتساب نتيجة الدالة وهي (٢) ثم ادخال (push) هذه النتيجة في المكدس .

٨- اخراج (pop) النتيجة الأخيرة وهي (٢) وكذلك المتغير (٣) مع عنوان الرجوع (Rt2) لاحتساب نتيجة الدالة وهي 2*3=6 ثم ادخال (push) هذه النتيجة في المكدس .

٩- اخراج (POP) النتيجة الأخيرة وهي (٦) وكذلك كل من المتغير (٤) مع عنوان الرجوع (Rt1) لاحتساب الدالة وهي 6*4=24 ثم ادخال (push) هذه النتيجة في المكدس .

١٠ – ان القيمة الوحيدة المتبقية في المكدس هي النتيجة النهائية لعملية احتساب مضروب العدد وهي 24 لأن 1*2*3*4= !n!=4

بسبب أسلوب المعالجة أعلاه نجد أن الاستدعاء الذاتي يتطلب مساحة خز نية اكبر (لانه خزن نسخه من قيم المتغيرات وعنوان الرجوع بعد كل استدعاء) كما أن هذه المعالجة تستغرق وقتا أطول ، ألا أننا نحصل على زيادة وضوح خوارزمية الحل وبساطته بما يساعد على سهولة الصيانة والتدقيق (verification & maintebility) .

ان إعداد برنامج فرعي بصيغة الاستدعاء الذاتي يتطلب مراعاة ما يأتي :

- أن يحتوي البرنامج الفرعي حالة (base case) وهي حالة توقف التكرار ، أي أنتهاء عمل البرنامج ، كما في برنامج حساب مضروب العدد (if (n<2) fact =1) .

- ان تنفيذ خطوات البرنامج الفرعي يؤدي الى اقتراب الحل من الوصول الى حالة (base case) .

ولو عدنا الى المثال الثاني فيمكن كتابة البرنامج الفرعي الدالة (power) كما يأتي :

I- تعريف الدالة بدون استخدام الاستدعاء الذاتي

```
int power(int x,int m)
{
 int p,i,power;
 p=1;
 if(m!=0)
 for(i=1;i<=m;i++)
 p*=x;
 power=p;
 return power;
}
```

II- تعريف الدالة باستخدام الاستدعاء الذاتي

```
int power1(int x,int m)
{
 int power;
 if(m==1)
 power=x;
 else
 power=x*power1(x,m-1);
 return power;
}
```

٥-٣ متى نلجا للاستدعاء الذاتي When We use Recursion

١- استخدام الاستدعاء الذاتي يفضل في العمليات التي يمكن تعريفها بصيغة التكرار الذاتي (تعريف نفسها بنفسها)

٢- استخدام الاستدعاء الذاتي يوفر الوقت والجهد للمبرمج عند الاعداد

٣- بصورة عامة يفضل الحل بدون الاستدعاء الذاتي اذا كان الحل قصيراً وبسيطاً .

٤- من التطبيقات المهمة التي يستخدم فيها الاستدعاء الذاتي هي (Tree Traversal، sorting، searching)

٥- في الاستدعاء الذاتي تستخدم صيغ وعبارات التفرع (if- else) , (switch-case) بدلاً من صيغ التكرار (for) (do-while) مع أهمية الاختبار الجيد للبرنامج وتدقيق قيم المتغيرات فيه قبل تنفيذه على البيانات الحقيقية .

تمرين : اكتب دالة استدعاء ذاتي لحساب القاسم المشترك الأعظم (Greatest Common Divisor) لأي عددين صحيحين موجبين (m, n) .

```
int GED(int m,int n)
{
 int r,ged;
 r=m%n;
```

```
 if(r==0)
 ged=n;
 else
 ged=GED(n,r);
 return(ged);
}
```

تمرين : سلسلة إعداد (Fibonacci) أي عدد فيها يكون مساوياً لمجموع العددين اللذين يسبقانه عدا العددين الأول =٠ والثاني =١ لذا فأن سلسلة الأعداد هي: (٠ ، ١، ١، ٢، ٣، ٥، ٨، ١٣، ٢١، ٣٤، ٥٥، ....)

```
int fib(int n)
{
 int fib1;
 if(n==0 || n==1)
 fib1=n;
 else
 fib1=fib(n-1)+fib(n-2);
 return fib1;
}
```

تمرين : اكتب برنامجاً فرعياً بصيغة الاستدعاء الذاتي لقراءة الرمز (space) وتجاوزه (اهماله)

```
void skipspaces()
{
 char ch[10];
 scanf("%c",&ch);
 if(ch==”space”)
 skipspaces();
}
```

تمرين :أكتب دالة بصيغة الاستدعاء الذاتي لاحتساب مجموع مربعات عناصر القائمة الموصولة (Start) .

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
struct node{
                int data;
                struct node*link;
                }*start,*p;
int sumsquares(struct node*start)
```

```
{
 int sum;
 if(start==NULL)
 sum=0;
 else
 sum=pow(start->data,2)+sumsquares(start->link);
 return sum;
}
```

برنامج –١١ :إيجاد أحد أعداد سلسلة أعداد (Fibonacci) في الموقع (N) باستخدام صيغة التكرار .

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
 int fnum1,fnum2,fn,n,i;
 clrscr();
 cout<<<"this program to generate the fibonacci numbers"<<endl;
 cout<<"of the form 0,1,1,2,3,5,8,13,21,34..."<<endl;
 cout<<"using iteration technique"<<endl;
 cout<<"--------------------------------------------"<<endl;
 cout<<"input the position of the number starting from position0:"<<endl;
 cin>>n;
 if(n<=1)
 cout<<n;
 else
 {
  fnum2=0;
  fnum1=1;
  for(i=2;i<=n;i++)
    {
     fn=fnum1+fnum2;
     fnum2=fnum1;
     fnum1=fn;
    }
   cout<<"the fibonacci number at position"<< n<<","<<fn;
  }
```

```
 getch();
}
```

برنامج –١٢ أيجاد أحد أعداد سلسلة أعداد (Fibonacci) في الموقع (N) باستخدام صيغة الاستدعاء الذاتي .

```
int fib2(int x)
{
 int n,fib;
 if(x==0 || x==1)
 fib=x;
 else
 fib=fib2(x-1)+fib2(x-2);
 return fib;
}
void main()
{
 int n;
 clrscr();
 cout<<"this program to generate the fibonacci numbers"<<endl;
 cout<<"of the form 0,1,1,2,3,5,8,13,21,34..."<<endl;
 cout<<"**************************************************"<<endl;
 cout<<"input the position of the number starting from position0:"<<endl;
 cin>>n;
 cout<<"the fibonacci no. at position "<<n<<"is "<<fib2(n);
 getch();
}
```

أسئلة الفصل الخامس

1- Write a recursive function that is given a string Sand returns the reversal of S. For example, given S=A B C D, the function should return the " DC B A ".

2- Writ a program to check if string (S) is the inverse of string (R) using stack.

3- What are the fundamental differences between recursion and iteration techniques?

4- Write a recursive procedure to search a list of nodes (names) and find the string "Target"


بسم الله

الفصل السادس

الأشجار Trees

1-6 المخطط : Graph

2-6 هيكل الشجرة : Tree Structure

3-6 الشجرة الثنائية: Binary Tree

4-6 انواع اخرى من الاشجار : Other types of trees

5-6 استعراض (مسح) عقد الشجرة: Tree Traversing

6-6 تمثيل الاشجار: Trees Representation

- الشجرة الاعتيادية: General Tree
- الشجرة الثنائية: Binary Tree

7-6 تمثيل التعابير الحسابية باستخدام الشجرة الثنائية:
Representation of arithmetic expressions using binary tree

8-6 تحويل الشجرة الاعتيادية الى شجرة ثنائية : Tree Transformation

9-6 شجرة البحث الثنائية : Binary Search Trees

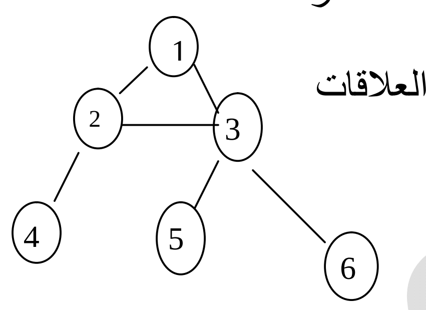
1-6 المخطط Graph

هو عبارة عن مجموعة من العناصر (V) تمثل بنقاط (رؤوس) تسمى (Vertices) ومفردها (Vertex) وهذه العناصر تربطها علاقات (E) تمثل بخطوط تسمى حافات (edges) ومفردها (edge) أي ان المخطط G =(V, E) هو مجموعة من العناصر والعلاقات وفق الشكل التالي :

{6,5,4,3,2,1}=V(G) العناصر

S(2,3),(1,3),(1,2)

(3,6),(3,5),(2,4) العلاقات



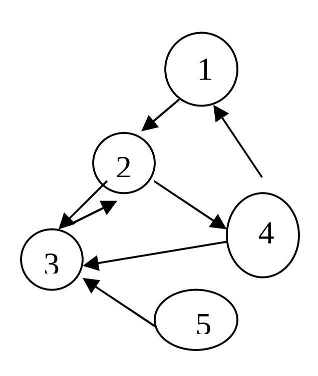
والمخططات على نوعين هما:

أ-المخطط غير المتجه undirected graph

هو المخطط الذي تكون العلاقة بين عناصره (رؤوسه) غير مرتبة (unordered) أي ان الاتجاه غير مهم في تلك العلاقة فمثلا الحافة (1,2) هي نفسها (2,1).

ب- المخطط المتجه directed graph

هو المخطط الذي تكون العلاقة بين عناصره (رؤوسه) مرتبة بنمط معين (ordered) أي ان الاتجاه مهم في تحديد تلك العلاقة فمثلا (1,2) تختلف عن (2,1) وتمثل هذه العلاقة بوضع سهم في مقدمة الخط ليوضح الاتجاه فالشكل (6-2) يبين أن هناك علاقة بين (3,2) ممثلة بمستقيم أي ان اتجاه العلاقة هي من ←3) وهناك علاقة اخرى تختلف عنها هي (2,3) ممثلة بمستقيم اخر ويعني ان العلاقة من ←2).



فمثلا لو كان المخطط اعلاه يمثل طرق المواصلات بين مجموعة المدن 5,4,3,2,1 فيمكن أن نقول أن هناك طريق من المدينة (2 ←) باتجاه واحد ولايسمح بأستخدامه من المدينة (← الى المدينة (1)ولكن هنالك طريق من المدينة (2)الى المدينة (3←)ويسمح باستخدامه باتجاه معكوس من المدينة (←(2) .

المسار path

هو مجموعة المستقيمات (الخطوط) التي تواصل بين أي نقطتين في المخطط فبين النقطتين 5,1 في الشكل الاول يكون المسار هو (3,1), (5,3).

طول المسار path length

يقصد به عدد المستقيمات (الخطوط) التي بربط او تصل بين اي نقطتين في المخطط فمثلا:
بين النقطتين٢ ٦ طول المسار=٢ وهما١(٢) ٢(٣)٣(٦) وطوله ٣ ومسار اخر هو ١(٣)٣(٦)وطوله ٢

المخطط المتصل connected graph

هو المخطط الذي توجد فيه مسارات بين اي نقطتين من نقاط المخطط

المخطط غير المتصل unconnected graph

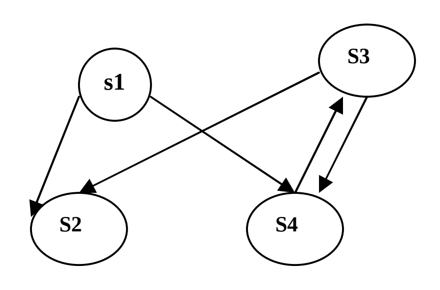
هو المخطط الذي تكون بعض نقاطه غير متصلة بمسار بينها

٦-١-١ تمثيل المخطط graph representation

ان اختيار طريقة تمثيل المخطط يعتمد على نوع التطبيق المطلوب انجازه وطبيعة وظائفه وسنوضح هنا طريقتين منها هما :

أ- استخدام مصفوفة المتجاورات adjacency matrix

يمثل المخطط بمصفوفة مربعة درجتها مساوية لعدد رؤوس (نقاط) المخطط (no. Of vertices) فاذا كان عدد الرؤوس (٣) فان المصفوفة تكون بابعاد (٣*٣) اما اذا كان عدد الرؤوس (٧)فان المصفوفة يجب ان تكون بابعاد (٧*٧) وهكذا بالنسبة للمخططات الاخرى :لناخذ المخطط التالي :



هذا المخطط هو مخطط متجه يتكون من (٤) نقاط رؤوس (vertices) (s1, s2,s3,s4) وخمسة خطوط (حافات-edges) ويمثل في مصفوفة مربعة درجتها (٤) وعناصرها (S i,j) حيث (i) يمثل نقطة البداية و(j) نقطة النهاية ففي حالة وجود خط (حافة) بين النقطتين يمثل الموقع بالقيمة (١) وبعكسه يمثل القيمة (٠) . فالصورة العامة للمصفوفة ستكون كالاتي :

S:	1	2	3	4
1	S 1,1	S 1,2	S 1,3	S 1,4
2	S 2,1	S 2,2	S 2,3	S 2,4
3	S 3,1	S 3,2	S 3,3	S 3,4
4	S 4,1	S 4,2	S 4,3	S 4,4

وعن تمثيل القيم لكل مسار ستصبح بالشكل التالي :

نقاط النهاية

S: (نقاط البداية)	1	2	3	4
1	0	1	0	1
2	0	0	0	0
3	0	1	0	1
4	0	0	1	0

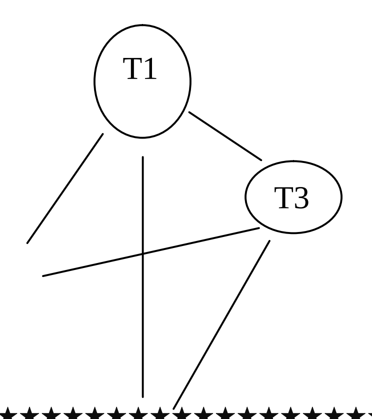
وهذه المصفوفة تعكس حالة المخطط اذ منها يتضح :-

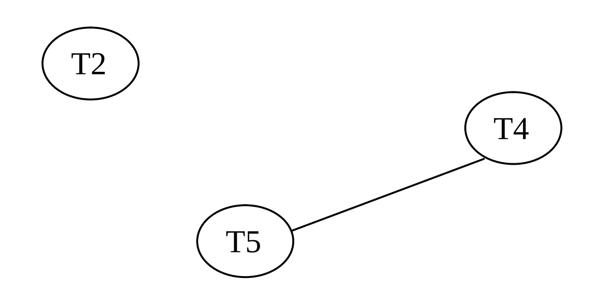
- وجود خط (حافة) من (S1 -- S2)،وجود خط (حافة) من (S1--S4)،وجود خط (حافة) من (S3--S2)،
- وجود خط (حافة) من (S3 --S4)،وجود خط (حافة) من (S4 --S3)،لايوجد خط من S3الى S1
- لايوجد خط من S2الى اية نقطة اخرى .لايوجد خط من S4الى S1او S2

ان المصفوفة التي تمثل المخطط المتجه تتصف بما يلي :-

- مجموع القيم في كل صف تعطي (تمثل) عدد الخطوط من كل نقطة فالصف الثالث (i=3) مثلا يكون مجموع القيم فيه هو (٢) لان النقطة الثالثة (s3) يخرج منها خطان الى كل من (s2,s4)
- مجموع القيم في كل عمود تعطي (تمثل) عدد الخطوط الداخلة (in degree) الى كل نقطة فالعمود الرابع (j=4) مثلا يكون مجموع القيم فيه هو (٢) لان النقطة الرابعة (s4) يدخل اليها خطان من (s1) ،(s3)

اما المخطط غير المتجه التالي :-





فيتكــون مــن (٥) نقــاط (رؤوس) هــي (T5,T4,T3,T2,T1) وســتة خطــوط (حافــات EDGES) يمثل في مصفوفة مربعة درجتها (٥) وتكون قيمة الموقع (١) في حالة وجود خط بين نقطتين بغض النظر عن الاتجاه فتكون المصفوفة كما في الشكل ويتضح فيها:

T:	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	0	0
3	1	1	0	0	1
4	0	0	0	0	1
5	1	0	1	1	0

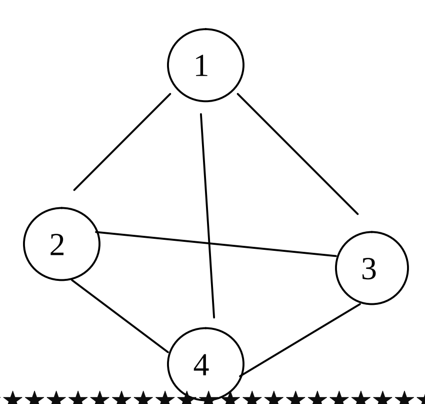
ان الخط الموجود من (T1) الى (T2) ممثل بالموقع (T(1,2 وقيمته (١) وهو نفس الخط الموجود من (T2) الى (T1) وممثل بالموقع (T(2,1 وقيمته (١) ايضا وهكذا بالنسبة للخطوط الاخرى بين اي نقطتين

ان هذه المصفوفة تتصف بما ياتي :

- متناظرة حول المحور (المثلث الاعلى يناظر المثلث الاسفل) ولهذا يمكن اختصار نصف المساحة الخزنية وذلك بتمثل احد المثلثين فقط
- ان مجموع القيم في كل صف (row) تعطي (تمثل) عدد الخطوط الخارجة (out degree) من كل نقطة فالصف الرابع (i=4) مثلا يكون مجموع القيم فيه هو (١) لان النقطة الرابعة (t4) يخرج منها خط واحد الى النقطة (t5).

ب – استخدام القوائم المتجاورة adjacency lists

تستخدم القائمة المتصلة (linked list) في تمثيل المخطط اذ ان كل عقدة من عقد المخطط تمثل بقائمة متصلة تحوي اسماء العقد التي تتصل بها فعناصر (عقد) القائمة الموصولة (i) هي الرؤوس المجاورة للعقدة (i) علما ان العقدة الواحدة تتالف من جزئين جزء يحتوي دليل الراس index of the vertex والجزء الاخر هو link مؤشر يشير الى موقع العقدة التالية .لناخذ المخطط غير المتجه في الشكل الاتي :-



يكون تمثيل هذا المخطط بقوائم متجاورة كل منها لها مؤشر رئيسي يشير الى بدايتها مثـل .... 2,vertex1 vertex الـخ وكمـا فـي الشـكل الاتـي

vetrex1	2	3	4 nil
vetrex2	1	3	4 nil
vetrex3	1	2	4 nil
vetrex4	2	2	2 nil

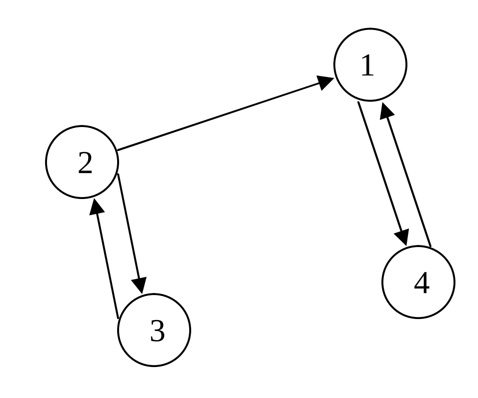
:

وتعرف برمجيا كالاتي :-

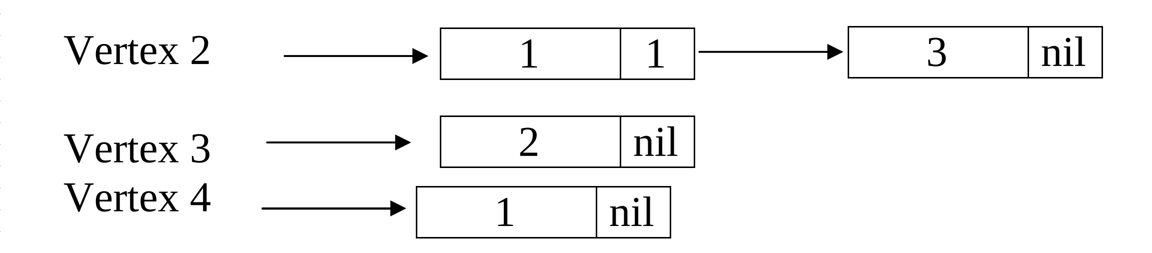
```
#include<iostream.h>
#include<conio.h>
const n=30;
struct node{
            int vertex;
            struct node*link;
           }*heads[n];
```

ولاستكمال عملية التمثيل هذه فان المؤشرات التي تشير الى بداية كل قائمة تخزن في مصفوفة احادية سعتها بقدر عدد القوائم او الرؤوس (n) (vertices) و (e) من الحافات (edges) وتمثيله يتطلب (2*e) من العقد ومصفوفة سعتها (n) لخزن المؤشرات الرئيسية التي تشير الى بداية كل قائمة وفي المثال اعلاه تجد اننا نحتاج الى (١٢) عقدة لان عدد الحافات هو (e=6) بالاضافة الى مصفوفة سعتها (٤) بقدر عدد الرؤوس

اما تمثيل المخطط المتجه في الشكل التالي فنلاحظ ان مجموع العقد في القوائم هو بقدر عدد الحافات في المخطط وهي (٥)



Vertex 1 ⟶ | 4 | nil |

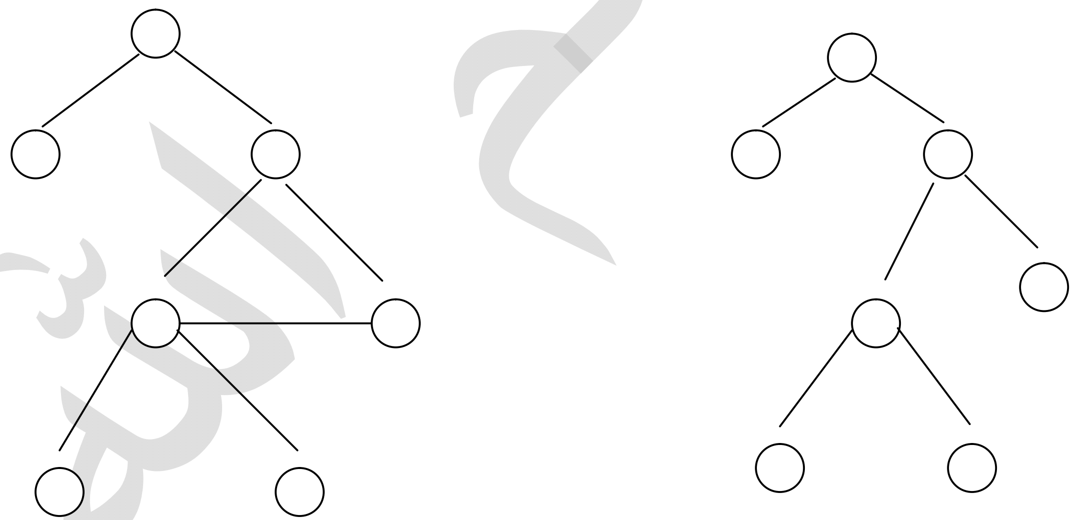


٦-٢ هيكل الشجرة Tree Structure

هنالك هياكل بيانية مماثلة للمخطط المتجه ،أي انها هياكل بيانية غير خطية (Non Linear) مثل تشعب طرق المواصلات في خرائط المدن، واعتمادا على مبادىء نظرية المخططات يساعد على تمثيل هذه الهياكل البيانية و التعامل معها من حيث البرمجة والتخزين باستخدام الحاسوب.

الشجرة Tree

هي تركيب من نوع مخطط متجه directed graph (Digraph) ، ولكن بدون تشكيل دائري (No cycle) اي ان هنالك خط واحد يوصل بين أي نقطتين (عقدتين).



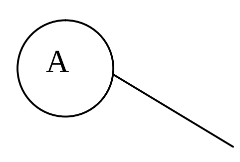
ليس هيكل شجرة لوجود التشكيل الدائري　　　　هيكل شجرة

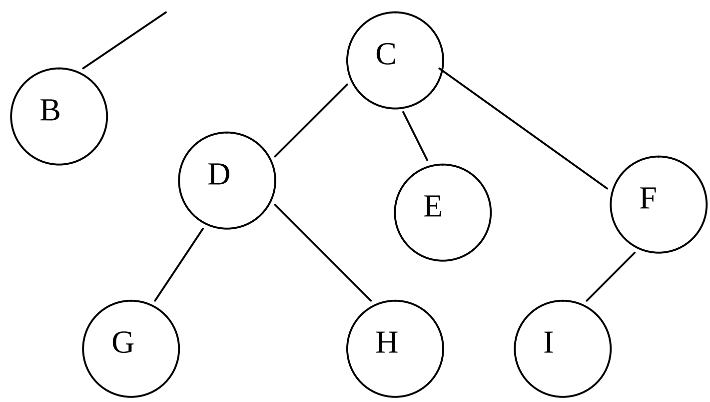
كما يمكن تعريف هيكل الشجرة بأنه مجموعة من العقد تتصف بما يأتي:-

+ توجد عقدة واحدة تسمى الجذر(Root) و هي التي لا يسبقها أية عقدة (العقدة التي ليس لها أب).

+ العقد المتبقية مجزأة الى مجموعات منفصلة كل منها هو هيكل شجرة أيضا يسمى شجرة فرعية(subtree).

لناخذ هيكل الشجرة الاتي :





وفيما يأتي عدد من التعريفات لتوضيح المفردات المتعلقة بهيكل الشجرة وطريقة استخدامها بالاشارة للشكل اعلاه

جذر الشجرة root

هوالعقدة التي لاتسبقها عقدة اخرى في الشجرة اي انها ليس لها اب اي العقدة (a) في الشكل السابق

العقدة المتفرع branched node

هي العقدة التي لها تفرع مثل F,D,C,A

العقدة النهائية (الورقة) Terminal (leaf) node

هي العقدة التي ليس لها تفرع مثل I,H,G,E,B (أي ليس لها أبناء).

مستوى العقدة Node level

هو عدد المسارات التي تبعد العقدة عن الجذر .

فمستوى عقدة الجذر = صفر، ومستوى العقدة E =٢، ومستوى العقدة H=٣

درجة العقدة NODE DEGREE

هي عدد المسارات الخارجة منها مباشرة (او عدد الابناء فيها) (او عدد التفرعات المباشرة منها)

فدرجة العقدة A=2 ودرجة العقدة f=1 ودرجة العقدة H=0 ودرجة العقدة C=3

درجة الشجرة tree degree

هي اعلى درجة من درجات العقد المكونة للشجرة فدرجة الشجرة المرسومة في الشكل اعلاه هي ٣

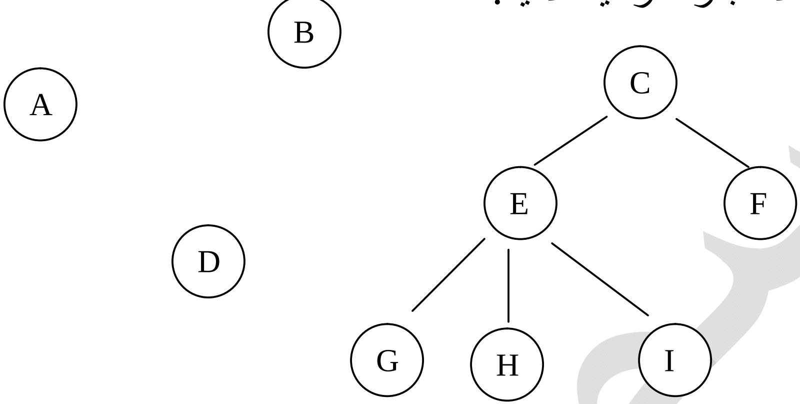
ارتفاع الشجرة هو اكبر مستوى (level) لاية عقدة في الشجرة اي هو اطول مسار في الشجرة

الشجرة الفرعية Subtree

يمكن تجزئة الشجرة الى اجزاء هي اشجار فرعية يكون لكل منها جذرا وقد تكون لها بعض الاوراق

شجرة فرعية

هذه الشجرة يمكن تجزئتها الى الاشجار الفرعية الاتية:



ويمكن الاستمرار في التجزئة الى ان نصل الى حالة تكون فيها كل عقدة شجرة فرعية :

ملاحظات اخرى :

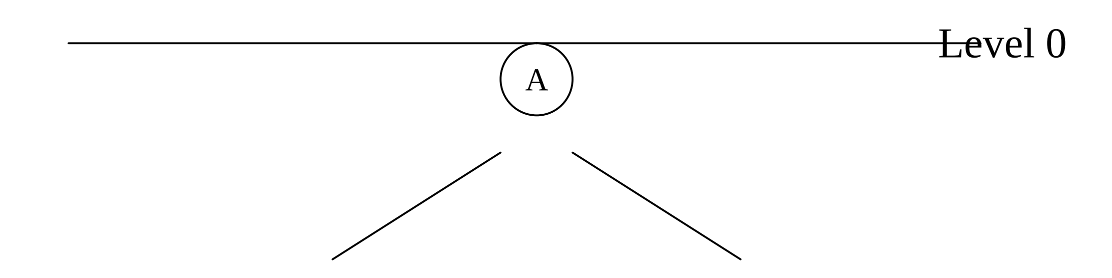
١- لايوجد توصيلات بين العقد في مستوى واحد

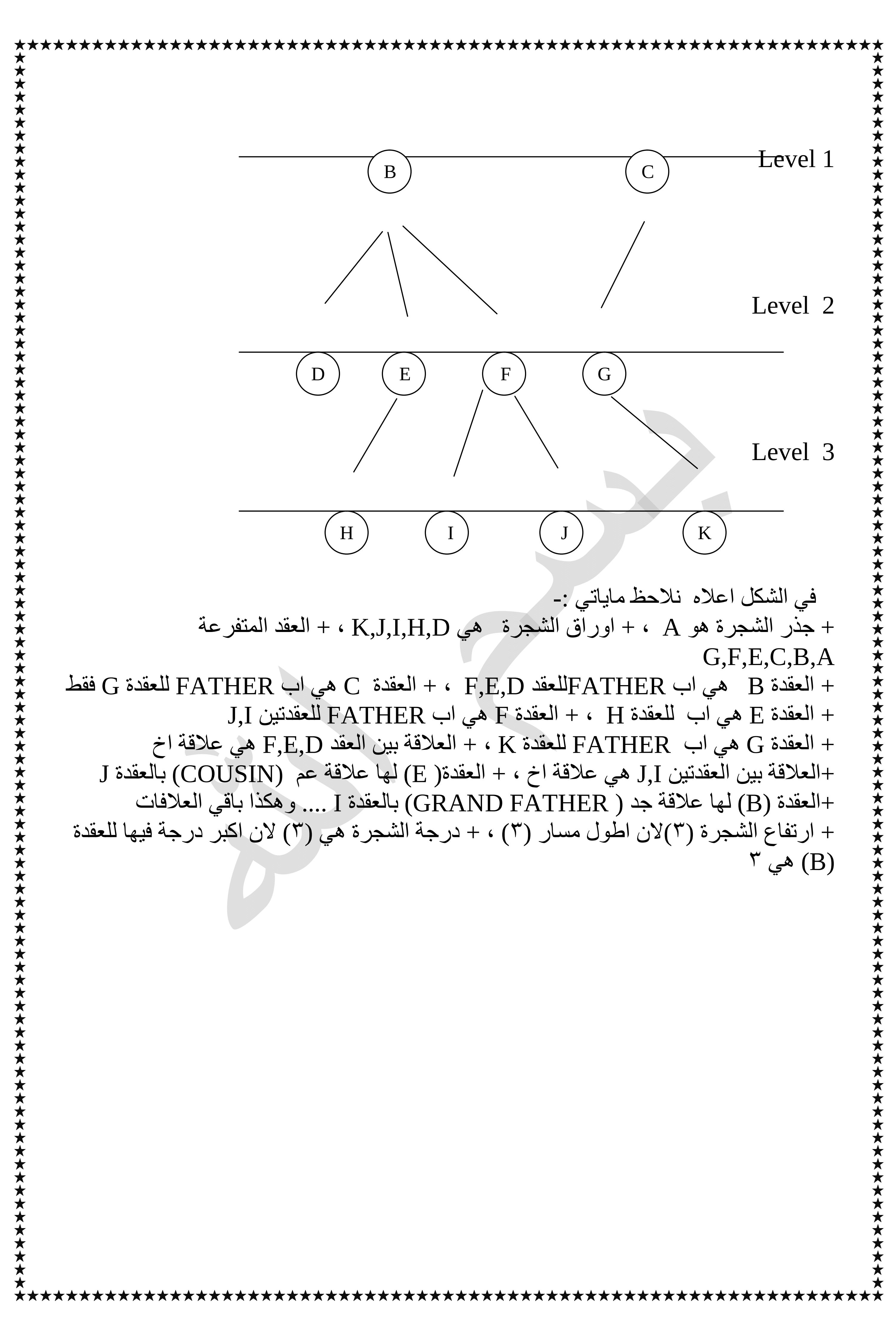
٢- لايوجد توصيلات بين اوراق الشجرة

٣- لاتوجد دوارات في الشجرة

٤- كل عقدة تعد بمثابة اب (father) بالنسبة للعقد المتفرعة منها مباشرة وكل من تلك العقد تكون بمثابة ابن (son)بالنسبة للعقدة الاب اي تستخدم المصطلحات العائلية (ابن – son –اخ brother - اب father – عم cousin – جد grand father – جد الجد grand of grand father) في تسمية العقد والعلاقات بينهما

٥- كل عقدة لها اب واحد one father





في الشكل اعلاه نلاحظ ماياتي :-

+ جذر الشجرة هو A ، + اوراق الشجرة هي K,J,I,H,D ، + العقد المتفرعة G,F,E,C,B,A

+ العقدة B هي اب FATHERللعقد F,E,D ، + العقدة C هي اب FATHER للعقدة G فقط

+ العقدة E هي اب للعقدة H ، + العقدة F هي اب FATHER للعقدتين J,I

+ العقدة G هي اب FATHER للعقدة K ، + العلاقة بين العقد F,E,D هي علاقة اخ

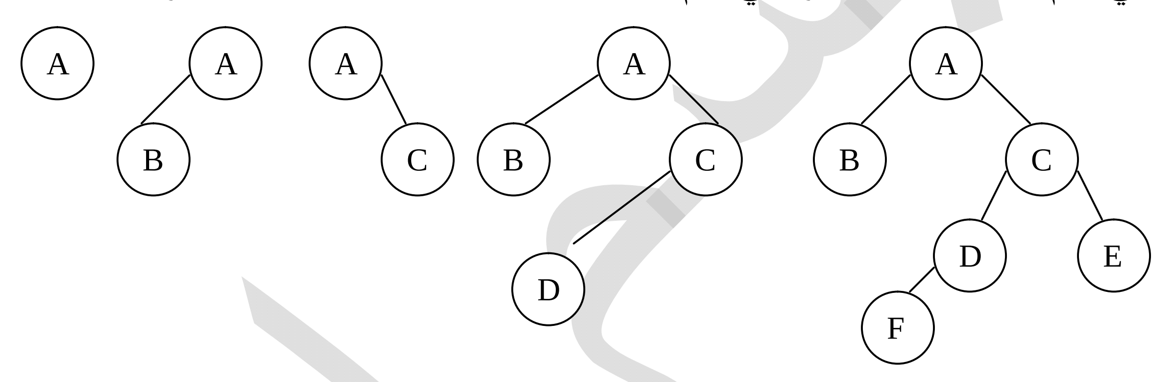
+العلاقة بين العقدتين J,I هي علاقة اخ ، + العقدة(E) لها علاقة عم (COUSIN) بالعقدة J

+العقدة (B) لها علاقة جد (GRAND FATHER) بالعقدة I .... وهكذا باقي العلافات

+ ارتفاع الشجرة (٣)لان اطول مسار (٣) ، + درجة الشجرة هي (٣) لان اكبر درجة فيها للعقدة (B) هي ٣

٦-٣ الشجرة الثنائية BINARY TREE

هي الشجرة التي كل عقدة فيها لاتحتوي على اكثر من عقدتين فرعين (AT MOST TWO SONS) اي ان درجة اي عقدة فيها لا تزيد على ٢ (اما ٠ او ١ او ٢) وهذه الشجرة تمثل هيكل بياتي مهم ولها تطبيقات كثيرة في علم الحاسبات و الاشكال الاتية تمثل اشجار ثنائية مختلفة



0
A
1
B
C
2
D
E
F
G
3
H
I

4

J

<u>اكبر عدد من العقد في المستوى</u>

في الشجرة الثنائية يكون اكبر عدد من العقد للمستوى(L) هو 2^L فالشجرة في الشكل السابق فان اكبر عدد ممكن من العقد فيه هو $2^2=4$ وهي G,F,E,D اما المستوى التالي (L=3) فان اكبر عدد ممكن من العقد هو
($2^3=8$) اي ان الحد الاقصى الذي لايمكن تجاوزه هو (٨) الاانه يمكن ان يكون اقل كما في الشجرة المذكورة حيث عدد عقد المستوى ٣ هي اثنان (I,H)

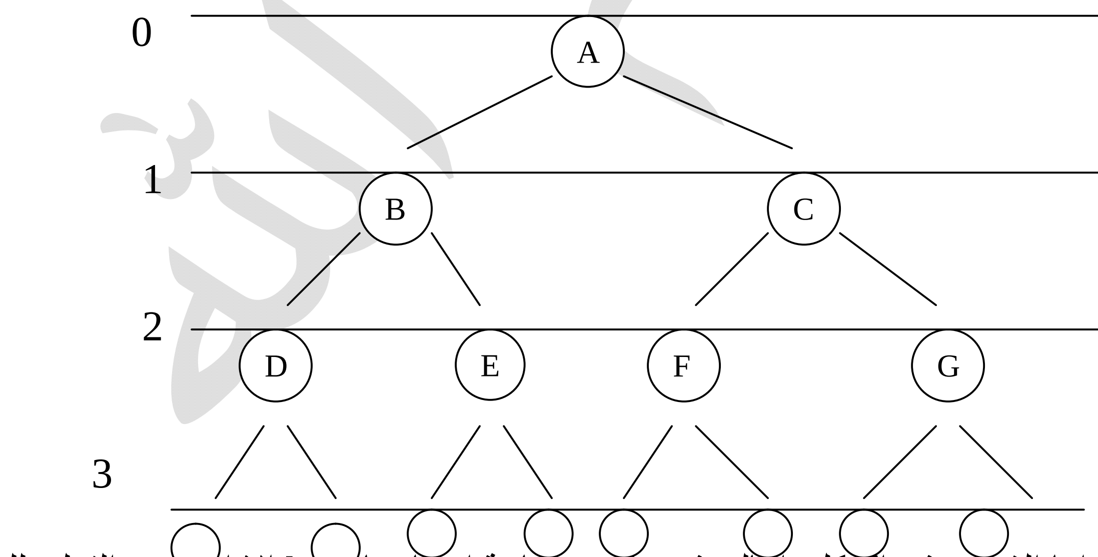
اكبر عدد من العقد في الشجرة الثنائية:

في الشجرة الثنائية التي ارتفاعها (H) فان الحد الاقصى لعدد العقد فيها هو $2^{(h+1)}-1$ وقد يكون العدد الفعلي للعقد اقل من هذا مثال :

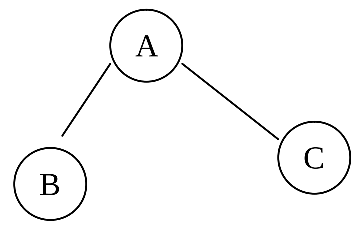
نلاحظ الشجرة الثنائية في الشكل ادناه ان ارتفاع h=3

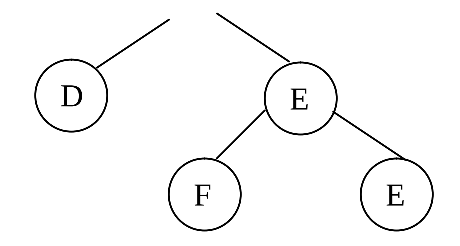
اكبر عدد من العقد فيها = ١- (١+ ٣)^ ٢

= ١-٢^٤=١-١٦=١٥



اما الشجرة في الشكل التالي فهي شجرة ثنائية ارتفاعها h=3 الا ان العدد الفعلي للعقد فيها هو (٧)





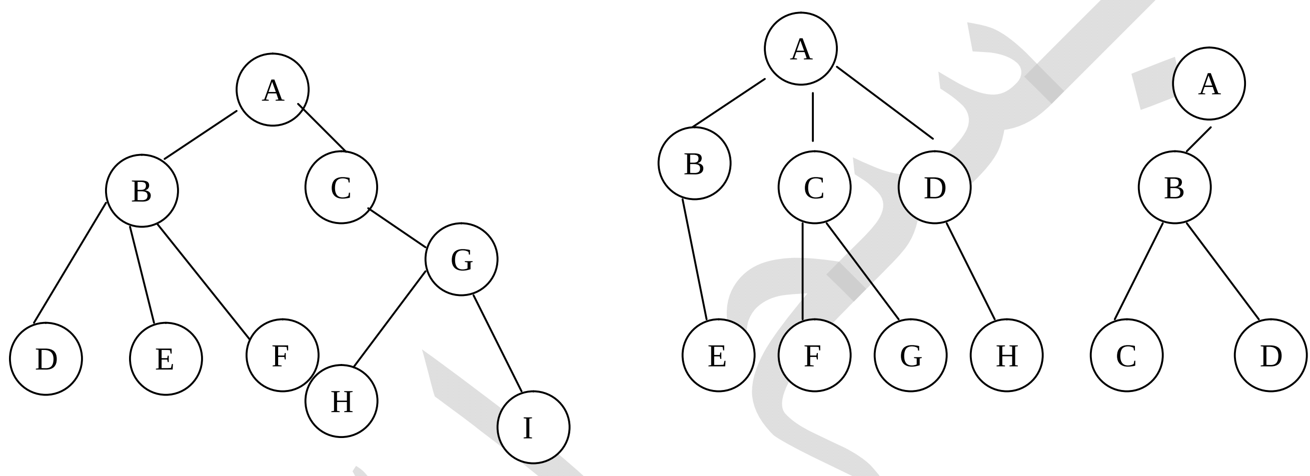
ملاحظات اخرى :-

- عدد اوراق الشجرة الثنائية = (عدد العقد التي درجتها ٢)+١، تكون الاشجار الثنائية متكافئة مع بعضها (equivalent)اذا كان لها نفس التركيب اي نفس الهيئة من حيث عدد ومواقع العقد وشكل التفرعات وتطابق البيانات

٦-٤ انواع اخرى من الاشجار :

الشجرة المتوازنة balanced tree

هي الشجرة التي جميع اوراقها تقع في مستوى واحد



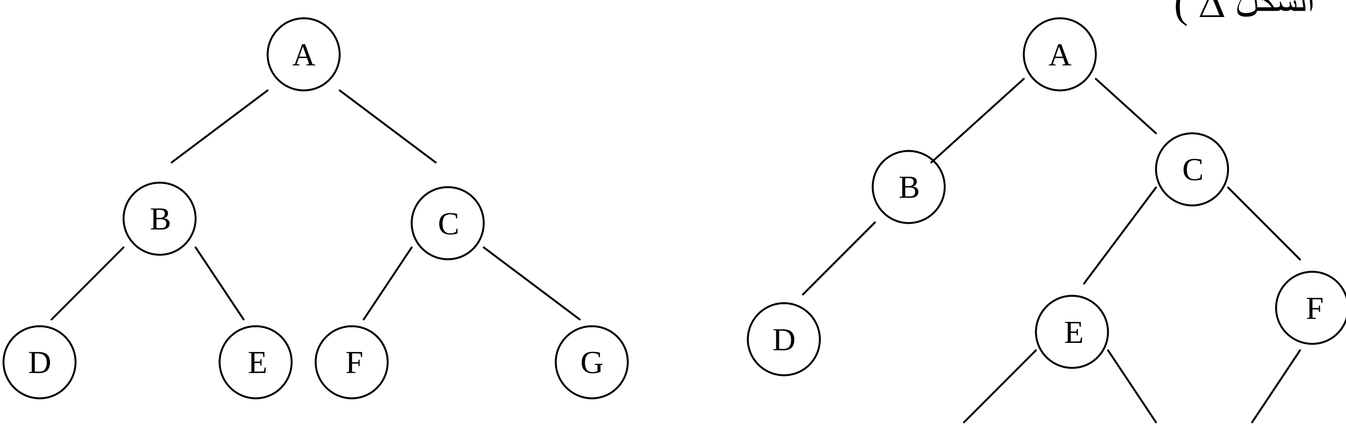
شجرة ثنائية متوازنة　　　شجرة متوازنة　　　شجرة غير متوازنة

الشجرة الثنائية المتوازنة balanced binary tree

هي الشجرة الثنائية التي اي عقدة فيها يكون لها فرعان

الشجرة الثنائية الممتلئة full binary tree

هي الشجرة التي جميع اوراقها في مستوى واحد واية عقدة متفرعه لها فرعان (تكون مثلثة الشكل Δ)



G H I J

شجرة ثنائية غير ممتلئة شجرة ثنائية ممتلئة

الشجرة الثنائية الكاملة Complete Binary Tree

هي الشجرة الثنائية التي تكون إما ممتلئة أو ممتلئة لحد المستوى قبل الأخير و تكون أوراق المستوى الأخير في أقصى اليسار.

A B A B A

ثنائية غير ممتلئة كاملة ثنائية غير ممتلئة و غير كاملة ثنائية و كاملة و ممتلئة

A B C D E A B C D A B C D E F G

ثنائية غير ممتلئة كاملة ثنائية غير ممتلئة و غير كاملة ثنائية كاملة و ممتلئة

ملاحظة: يكون شكل الشجرة مثلث إذا كانت ممتلئة أو يكون شكلها

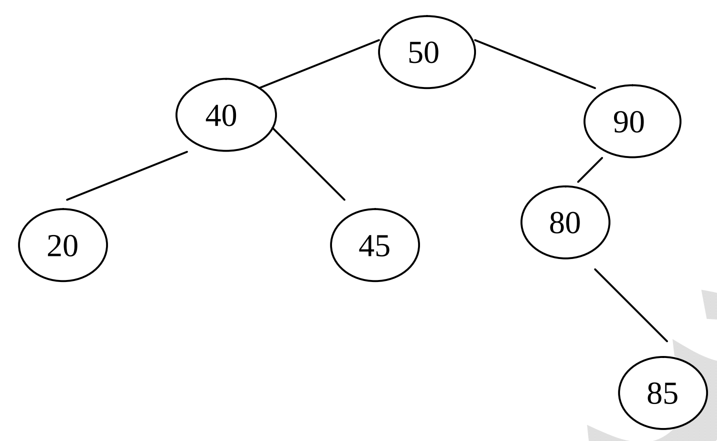
شجرة AVL-Tree

وهي شجرة ثنائية غيرخالية والفرق في ارتفاع الشجرة لأية عقدة فيها لا يزيدعلى١،أي أن ارتفاع الشجرة الفرعية اليسرى(TL) لأية عقدة لا يزيد على ارتفاع الشجرة الفرعية اليمنى (TR) بأكثرمن١أي أن $hL- hR =<1$

ملاحظة :اسم الشجرة مختصر لاسماء الأشخاص الثلاثة الذين استخدموها :– Adelson Velskii- Landis

شجرة البحث الثنائية Binary Search Tree

هي الشجرة الثنائية التي تكون قيمة عنصر الفرع الايسر (الابن) لأية عقدة هي اقل من قيمة عنصر تلك العقدة باعتبارها الاب(father) وتكون قيمة عنصر الفرع الايمن (الابن) اكبر من قيمة عنصر العقدة (الأب).



الشجرة m-way search Tree

هي شجرة بحث متوازنة تكون جميع عقدها بدرجة (m) او اقل.

الشجرة B-Tree

هي شجرة بحث بدرجة(m)وتكون اما خالية او ارتفاعها =>١. وتتوفر فيها الحقائق التالية:

- عقدة الجذر لها فرعان على الاقل
- جميع العقد الاخرى (عدا الجذر والاوراق) تكون درجتها على الاقل (m/2) .
- جميع الاوراق تنتهي في مستوى واحد.

ملاحظة: اسم الشجرة ورد من اسم الشخص الذي استخدمها وهو (bayer)

6-15 استعراض (مسح) عقدا لشجرة Tree Traversing

ان عملية المسح تعني المرور(زيارة visit) كل عقدة في الشجرة مرة واحدة فقط ولا يجوز تكرار الزيارة.

وبالنظر لكون هيكل الشجرة هو هيكل بياني لاخطي لذا فأن عمليات البحث عن عنصر(عقدة) معين في هذا الهيكل او اضافة عنصر إليه او حذف عنصر منه تختلف عن اسلوب التعامل مع الهياكل الاخرى وان اختيار إحدى هذه الطرق يعتمد على كيفية تمثيل الشجرة في الذاكرة. وفيما يأتي اهم الطرق المستخدمة لهذا الغرض:

اولا- الاستعراض حسب المستويات Level by Traversing

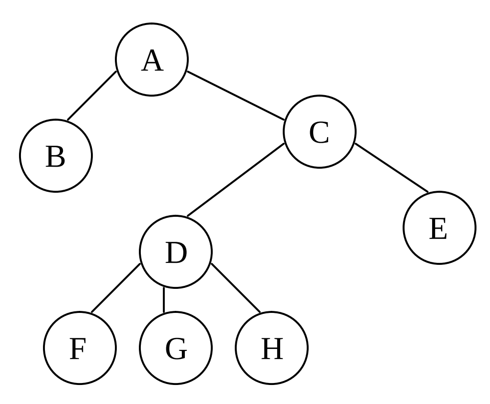
الاستعراض من أعلى إلى أسفل Top–down Traversing

وتتلخص الخوارزمية بالخطوات التالية:

1- البدء بعقدة الجذر..

2- استعراض عقد المستوى التالي ومن اقصى اليسار الى اليمين.
3- الاستمرار بنفس الطريقة بالانتقال الى المستويات الادنى بالتتابع بدءا بالعقدة في اقصى اليسار ثم اليمين .

<u>مثال</u>: لناخذ الشجرة الآتية :



تكون نتيجة استعراض عقدها بطريقة Top-Down هي: A B C D E F G H

<u>ب/الاستعراض من اسفل الى اعلى Bottom- Up Traversing</u>

1- البدء بالورقة في اقصى بأدنى مستوى .
2- التحرك نحو العقدة في اليمين منها وبنفس المستوى لحين الانتهاء من زيادة جميع عقد ذلك المستوى .
3- الانتقال الى المستوى الاعلى وزيادة العقد فيه أيضا من اليسار الى اليمين ،وهكذا تستمر العملية لحين الوصول الى جذر الشجرة .

اى ان نتجية استعراض نفس الشجرة بهذه الطريقة تكون :-   F G H D E B C A

<u>ملاحظة</u>:نرى ان التعامل مع الشجرة واجزائها يوضح ان التشكل الاساسي والمتكرر فيها هو تكونها من عقدة الجذر (N).وقد تحتوي ورقة او اكثر او بدون اورق لذا فالتعامل معها يمكن ان يبدا بالجذر ولنفرضه (N) او بالورقة في اقصى اليسار (L)او بالورقة في اقصى اليمين (R) .

أي ان الشكل العام للشجرة

N
L   R

ولهذا فأن احتمالات الاستعراض هي ستة   RLN RNL LRN LNR NRL NLR

وسناخذ فقط الحالات التي تمثل الاستعراض من اليسار L الى اليمين R وهي ثلاثة:

<u>NLR</u> أي البدء بالجذر (N) ثم التحرك نحو اليسار (L) ثم اليمين (R) ولكون الجذر يذكر هنا مسيقا تسمى هذه الطريقة بالترتيب السابق (preorder بالترتيب السابق (preorder) .

<u>LRN</u> أي البدء باليسار (L) ثم اليمين (R) والانتهاء بالجذر (N) أي أن ذكر الجذر يأتي لاحقا وتسمى هذه الطريقة بالترتيب اللاحق (post order) نسبة الى الجذر (N).

<u>LNR</u> البدء باليسار (L) ثم الجذر (N) ويأتيه اليمين(R) أي ان الجذر يأتي في الوسط وتمسى

هذه الطريقة (inorder) نسبة الى الجذر (N) .

ثانياً : الاستعراض بالترتيب السابق Preorder Traversing (NLR)

وتتلخص خطوات هذه الخوارزمية بالآتي :

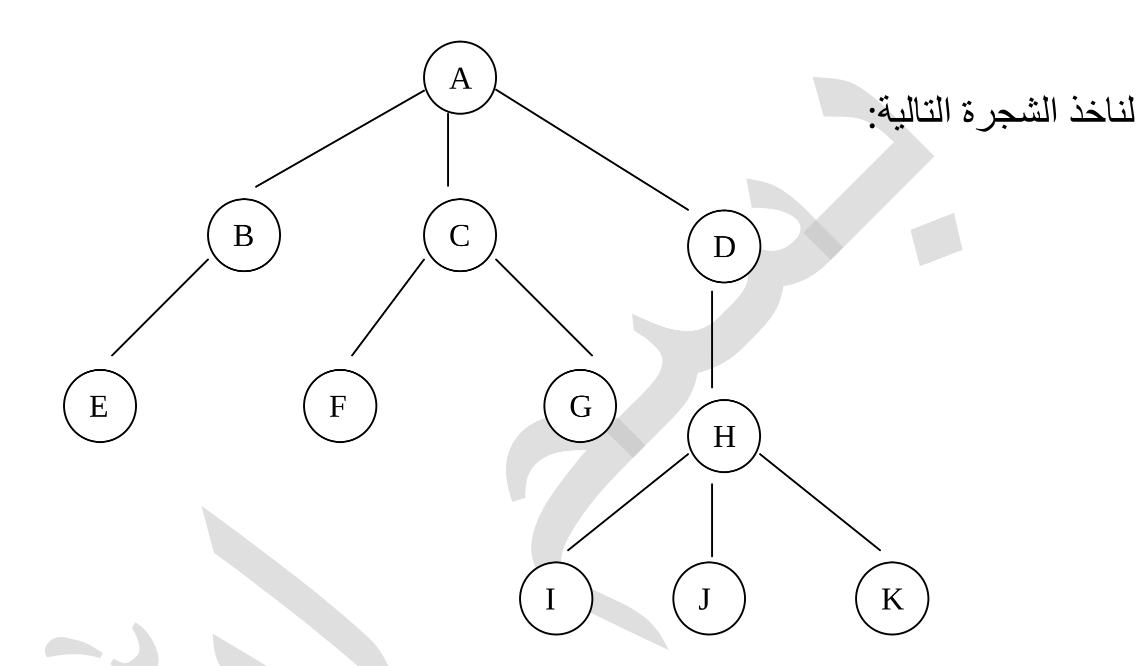
1-البدء بعقدة الجذر (N)

2-استعراض الشجرة الفرعية في اقصى اليسار .

3-داخل الشجرة الفرعية يتم الاستعراض من اقصى اليسار (يمثل اكبر الابناء)ثم التحرك لليمين .

4-في حالة لايوجد فرع في اليمين (لايوجد اخ brother) يكون الانتقال الى العم father s (brother)

لناخذ الشجرة التالية:



ستكون نتيجة الاستعراض كالآتي :- A B E C F G D H I J K

في هذه الطريقة نلاحظ مايأتي :

1- جميع الاباء يذكرون قبل الابناء A قبل D, C, B

E قبل G , F

H قبل K, J, I

وهكذ ....

2- لو مثلنا هذا الاستعراض بالسير حول الشجرة بخط متقطع لوجدنا أن العقدة تذكر عند اول مرور بها بدأ من الجذر ..

3-تستخدم هذه الطريقة لتمثيل التعابير الحسابية بصيغة Polish Notation

ثالثا: الاستعراض بالترتيب اللاحق Post order Traversing (LRN)

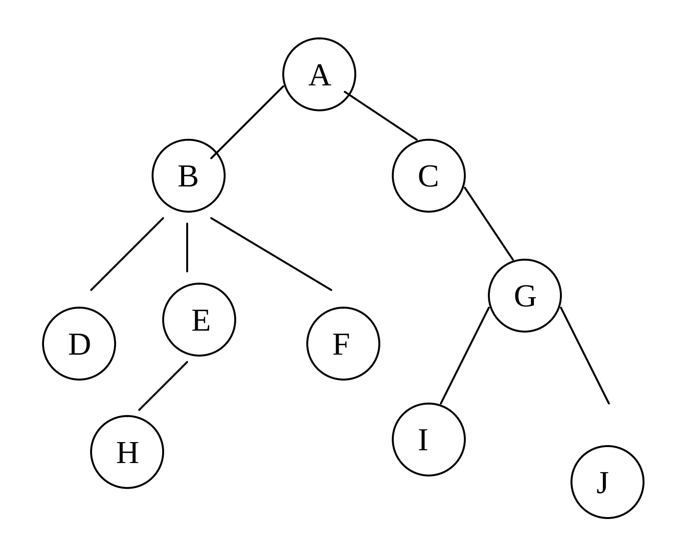
تتلخص خطوات هذه الخوارزمية بالآتي :-

1-البدء بالعقدة الورقة في اقصى يسار الشجرة ثم الاوراق التي على يمينها (ان وجدت).

2-الانتقال الى العقدة الاعلى (father) (أي اب تلك العقدة).

3-مسح الشجرة الفرعية التالية في اليمين بنفس الطريقة لحين الوصول الى الجذر.

لناخذالشجرة التالية :-



وستكون نتيجة الاستعراض كالآتي :   D   H   E   F   B I   J   G   C   A

في هذه الطريقة نلاحظ مايأتي :-

١-جميع الاباء يذكرون بعد الابناء فمثلا B بعد D, E, F - E بعد H - G بعد I J, - وهكذا ...

٢ – لو مثلنا هذا الاستعراض (المسح) بالسير حول الشجرة (الخط المنقط)لوجدنا ان العقدة تذكر بعد مغادرتها بدء من الورقة في أقصى اليسار وانتهاء بالجذر .

٣ – تسخدم هذه الطريقة لتمثيل التعابير الحسابية بصيغة Reverse Polish Notation(RPN)

رابعا : الاستعراض بالترتيب البيني Intruder Traversing

ان هذه الطريقة تستخدم في مسح الاشجار الثنائية فقط وتتلخص خطوات الخوارزمية فيها بالآتي :

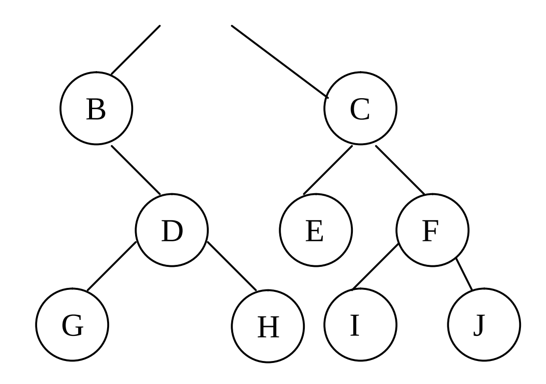
١-البدء بالعقدة الورقة في اقصى يسار الشجرة .

٢-الانتقال الى عقدة الجذر (اب تلك العقدة) .

٣- زيادة العقدة التي في اليمين (ان وجدت) وفي حالة عدم وجودها الانتقال الى الجد (grand father) .

لناخذ الشجرة الثنائية الآتية :

A



ان نتيجة استعراض (مسح)هذه الشجرة هي : B G D H A E C I F J

وفي هذه الطريقة نلاحظ ماياتي :-

١- لو مثلنا هذا الاستعراض بالسير حول الشجرة (خط المنقط) لوجدنا ان العقدة تذكر عند المرور تحتها .

٢- تستخدم هذه الطريقة لتمثيل التعابير الحسابية بصيغة Infix Notation

٦-٦ تمثيل الأشجار Tree Representation

هنالك عدة طرق لتمثيل الأشجار عند الخزن في الحاسوب وتحديد أفضلها يعتمد على :

+ العمليات التي تتطلبها المسالة المعينة بالحل

+ منظومة الحاسوب المستخدمة .

+ لغة البرمجة .

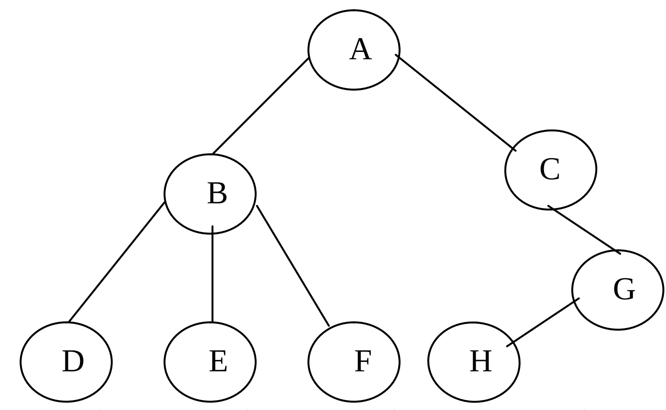
٦-٦-١ تمثيل الاشجار الاعتيادية General Tree Representation

أ/ عدد المؤشرات بقدر اكبر عدد من الفروع

لكل عقدة في الشجرة عدد معين من الابناء different No . of children وباستخدام القائمة الموصولة (Linked list)لتمثيل مثل هذه الشجرة فيجب تحديد مؤشر (pointer) لكل ابن child ولهذا فالعقدة التي لها ابن واحد (one child) تحتاج الى مؤشر واحد .والعقدة التي لها ابنان (two children) تحتاج الى مؤشرين ...... وهكذا

وهذه القائمة الموصولة يجب ان تعرف فيها عدد المؤشرات بقدر اكبر عدد من الابناء لأية عقدة في الشجرة ، وهذا يعني ان كل عقدة سيكون لها نفس العدد من المؤشرات حتى لو كان عدد فروعها (أبنائها) اقل من ذلك ، وهذا سيعني ضياع كبير في استخدام المساحة الخزنية .

لناخذ الشجرة التالية :



نلاحظ ان عدد التفرعات من كل عقدة يختلف واكبر التفرعات هو ثلاثة للعقدة B اذ لها ثلاثة ابناء F,E,D ويكون تعريف جميع عناصر القائمة الموصولة بعدد مؤشرات يساوي (٣) لكل عنصر وكالآتي :-

```
struct node{
        int data;    /*or any type*/
        struct node*ptr1,*ptr2,*ptr3;
        }*tree;
```

ويمكن تمثيل ذلك بالرسم التالي

Tree

A | L1 | L2 | L3

B | L1 | L2 | L3

C | L1 | L2 | L3

D | L1 | L2 | L3

E | L1 | L2 | L3

F | L1 | L2 | L3

G | L1 | L2 | L3

H | L1 | L2 | L3

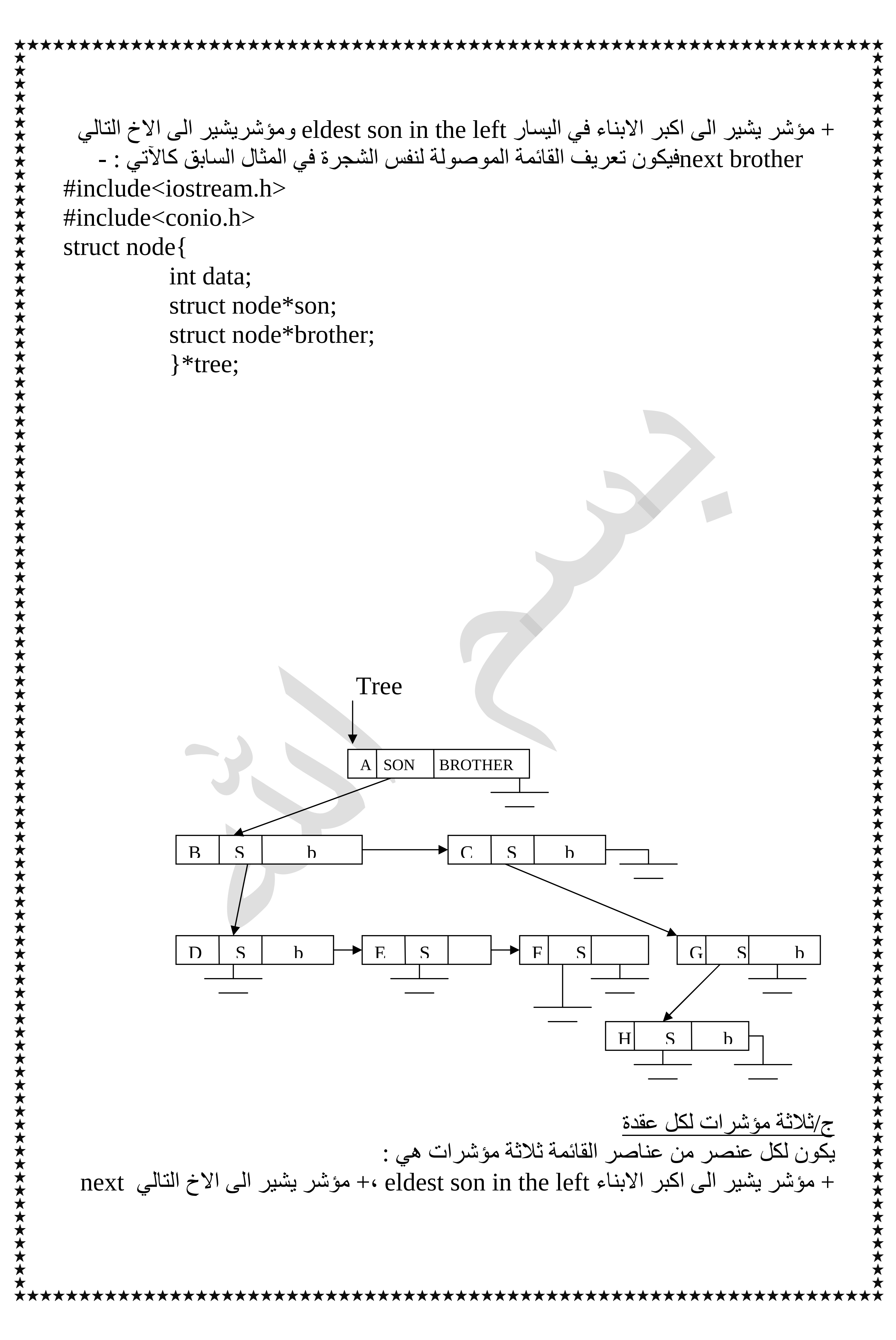
لاحظ ان المؤشر الزائد في العقدة يؤشر nil

ب/ مؤشرين لكل عقدة

يكون لكل عنصر من عناصر القائمة مؤشرين هما:

+ مؤشر يشير الى اكبر الابناء في اليسار eldest son in the left ومؤشريشير الى الاخ التالي next brotherفيكون تعريف القائمة الموصولة لنفس الشجرة في المثال السابق كالآتي : -

```
#include<iostream.h>
#include<conio.h>
struct node{
        int data;
        struct node*son;
        struct node*brother;
        }*tree;
```



ج/ثلاثة مؤشرات لكل عقدة

يكون لكل عنصر من عناصر القائمة ثلاثة مؤشرات هي :

+ مؤشر يشير الى اكبر الابناء eldest son in the left ،+ مؤشر يشير الى الاخ التالي next

brother

+ مؤشر يشير الى الاب nodes father

وعليه يكون تعريف القائمة الموصولة لنفس الشجرة في المثال السابق كالاتي:

```
 #include<iostream.h>
#include<conio.h>
struct node{
        char data;
        struct node*son;
        struct node*brother;
        struct node*father;
          }*tree;
```

٦-٦-٣ تمثيل الاشجار الثنائية Binary tree Representación

أ-استخدام المصفوفة Array Representation

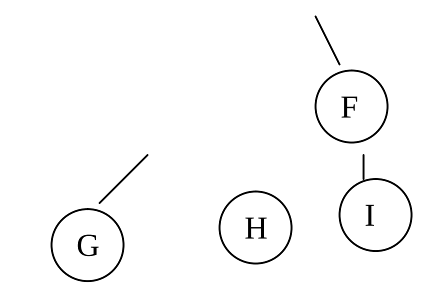
تستخدم مصفوفة احادية بسعة مساوية لأكبر عدد ممكن لعقد الشجرة الثنائية التي ارتفاعها (h) باعتماد العلاقة (1 -2) وتخزن القيم البيانية للعقد وفق الاتي:-

١- تخزن عقدة الجذر في الموقع الاول من المصفوفة وليكن T[1] .

٢- تخزن العقد الأخرى بحيث :-

- عقد الابن الايسر (left child) للعقدة في الموقع (I) تكون في الموقع (١*٢).
- عقدة الابن الايمن (Right child) في الموقع (I) تكون في الموقع (١+ ١ *٢).
- ان عقدة الاب لأية عقدة في الموقع (I) تكون في الموقع (I DIV 2)

مثال :لناخذ الشجرة التالية :



بما ان ارتفاع الشجرة h=3 لذا فأن اكبر عدد ممكن من العقد في مثل هذه الشجرة سيكون $=2^{(h+1)}-1$

$=2^{(3+1)}-1=16-1=15$

اذن سعة المصفوفة لتمثيل هذه الشجرة هو (١٥)

ولتكن المصفوفة هي (١٥) T

T(15):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ان عقدة الجذر (A) تخزن في الموقع الاول T(1) .
- والعقدة (b) هي الابن الايسر للعقدة (A) تكون في الموقع T(2) لان $2*I=2*1=2$.
- والعقدة (C) هي الابن الايمن للعقدة (A) تكون في الموقع T(3) لان $2*I+2*1+1=3$.
- العقدة(D)هي الابن الايسر للعقدة B تكون في الموقع T(4) لان$4=2*2=2*1$ وهكذا تحدد مواقع باقي العقد.

ملاحظة: بعد توزيع العقد في المصفوفة يمكن ان نلاحظ ان الوصول الى العقدة الاب لايه عقدة مثل G التي هي في الموقع T(10) تكون في الموقع T(5) اي القدة E لان I DIV 2 = 10 DIV 2 =5

ب- استخدام القيد record representation

في هذه الطريقة تستخدم القائمة الموصولة (LINKED LIST) لتمثيل الشجرة وبعدة اساليب هي:

١- تمثيل العقدة بمؤشرين حيث يشير احدهما للابن الايسر(L Ch) والاخر يشير الى الابن الايمن (R Ch) .

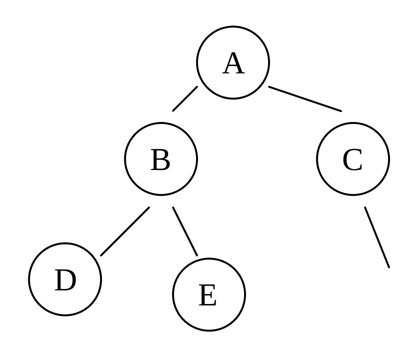
```
#include<iostream.h>
#include<conio.h>
struct node{
        char data;   /*or any other type*/
        struct node*Lchild;
        struct node*Rchild;
        };
```

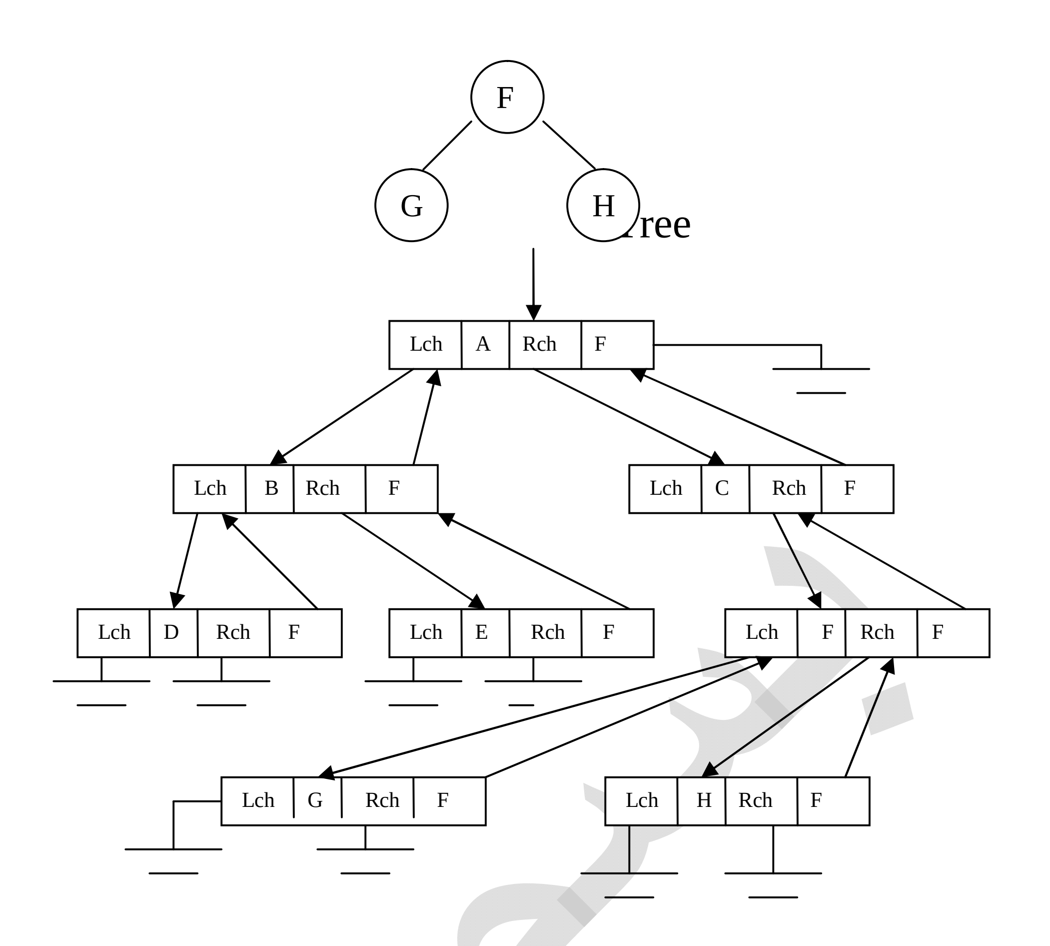
وفيما ياتي تمثيل الشجرة الثنائية الاتية:

٢- تمثيل العقدة بثلاث مؤشرات (THREE POINTERS) تعرف كل عقدة لتحتوي على مؤشر يشير الى الابن الايسر ومؤشر يشير الى الابن الايمن ومؤشر ثالث يشير الى العقدة الاب.

```
#include<iostream.h>
#include<conio.h>
struct node{
            int data;      /*or any other type*/
            struct node*Lch;   /*left child ptr*/
            struct node*Rch;  /*right child ptr*/
            struct node*f;       /*father ptr*/
           };
```

فيما ياتي تمثيل الشجرة الثنائية التالية:





البرامج الفرعية للشجرة الثنائية

سبق ان ذكرنا ان هيكل الشجرة هو من الهياكل التي اجزاءها (SUBTREES) اشجارا ايضا اي ان الجزء يشبه الكل من حيث ان الشجرة الفرعية هي شجرة ايضا وهنا نستطيع الاستفادة من صيغة الاستدعاء الذاتي (RECURSION) في كتابة البرامج الفرعية لتمثيل الشجرة والعمليات عليها كالاتي :-

برنامج فرعي لاستعراض الشجرة الثنائية بطريقة الترتيب السابق:

```
void preorder(struct node*root)
{
 if(root!=NULL)
  {
   cout<<endl<<root->data<<endl;
   preorder(root->Llink);
   preorder(root->Rlink);
  }
}
```

من الملاحظ ان هذا البرنامج الفرعي يعكس خطوات الخوارزمية المتمثلة بمسح (الجذر/الفرع الايمن) وتكرار هذه الخطوات عند كل عقدة باعتبارها شجرة فرعية ولهذا فانه برنامج فرعي ذاتي التكرار (Recursive)

برنامج فرعي (procedure) لاستعراض الشجرة الثنائية بطريقة الترتيب اللاحق

```
void postorder(struct node*root)
{
 if(root!=NULL)
  {
   postorder(root->Llink);
   postorder(root->Rlink);
   cout<<endl<<root->data<<endl;
  }
}
```

ان هذا البرنامج الفرعي هو (Recursive procedure) لتمثيل خطوات الخوارزمية (الفرع الايسر/ الجذر) المتكررة عند كل عقدة(باعتبارها شجرة فرعية)

برنامج فرعي (procedure) لاستعراض الشجرة الثنائية بطريقة الترتيب البيني

```
void inorder(struct node*root)
{
 if(root!=NULL)
  {
   inorder(root->Llink);
   cout<<endl<<root->data<<endl;
   inorder(root->Rlink);
   }
}
```

وهذا البرنامج هو ايضا من نوع (recursive procedure) لتمثيل الخطوات الخوارزمية (الفرع الايسر /الجذر/ الفرع الايمن) المتكررة عند كل عقدة (باعتبارها شجرة فرعية)

برنامج فرعي :بصيغة التكرار لاستعراض عقد الشجرة الثنائية بطريقة الترتيب البيني(Inorder Traversing)

```
void non_recinorder(struct node*root)
{
 int top;
 struct node*p;
 clearstack();
 p=root;
 do
  {
   while(p!=NULL)
     {
      push(p->data);
```

```
      p=p->left;
     }
if(!emptystack())
   {
    pop();
    cout<<endl<<p->data<<endl;
    p=p->right;
   }
   }while(p!=NULL&&!emptystack());
}
```

برنامج فرعي :- بصيغة الاستدعاء الذاتي لاحتساب عدد الاوراق في الشجرة الثنائية (no. of leaves)

```
void leaves(struct node*r)
{
 int count=0;
 if(r!=NULL)
  {
   if((r->left==NULL)&&(r->right==NULL))
   count++;
   leaves(r->left);
   leaves(r->right);
  }
  cout<<"the number of leaves="<<endl<<count;
}
```

برنامج فرعي : بصيغة الاستدعاء الذاتي لمبادلة (swap) قيمة العنصر في الفرع الايسر مع قيمة العنصر في الفرع الايسر مع الفرع الايسر مع قيمة العنصر في الفرع الايمن لكل عقدة في الشجرة الثنائية

```
void swapnodes(struct node*r)
{
 struct node*t;
 if(r!=NULL)
  {
   t=r->left;
   r->left=r->right;
   r->right=t;
   swapnodes(r->left);
   swapnodes(r->right);
```

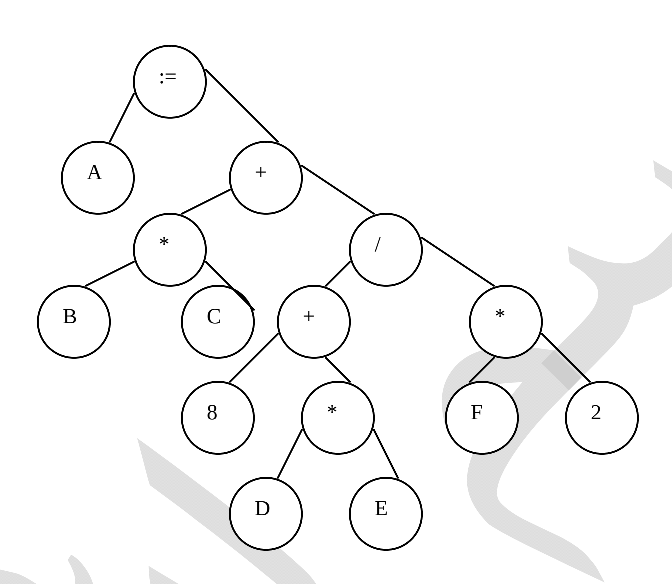
}
}

٦-٧ تمثيل التعابير الحسابية باستخدام الشجرة الثنائية

Representation of Arithmetic Expressions using Binary Tree

من التطبيقات المهمة للاشجار الثنائية هو استخدامها في تمثيل التعبير الحسابي اذ ان العملية الحسابية (+، -،*،/ ..... الخ)تمثل بعقدة متفرعة اما العوامل الحسابية فتمثل بالاوراق مع ملاحظة ان مستويات الشجرة تعكس اسبقيات تنفيذ العمليات الحسابية في ذلك التعبير الحسابي

مثال : استخدام الشجرة الثنائية لتمثيل التعبير الحسابي A:=B*C+(8+D*E) / (F*2)



عند استعراض (مسح) هذه الشجرة بكل من :-

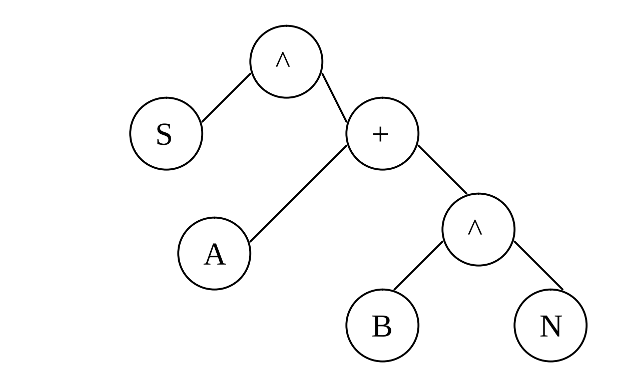
١- طريقة الترتيب البيني (INORDER) فنحصل على التعبير الحسابي نفسه وهو بصيغة (Infix notation)

٢- طريقة الترتيب السابق ونحصل على صيغة (prefix notation) للتعبير الحسابي اي :-
:= A+ * BC/+8*DE*F2 ، وتسمى ايضا صيغة(Polish Notation)

٣– طريقة الترتيب اللاحق ونحصل على صيغة (postfix notation) للتعبير الحسابي أي
ABC *8DE * + F 2 */:= ، وتسمى ايضا صيغة (Reverse Polish Notation)

تمرين :استخدام الشجرة الثنائية لتمثيل كل من التعابير الحسابية التالية :-

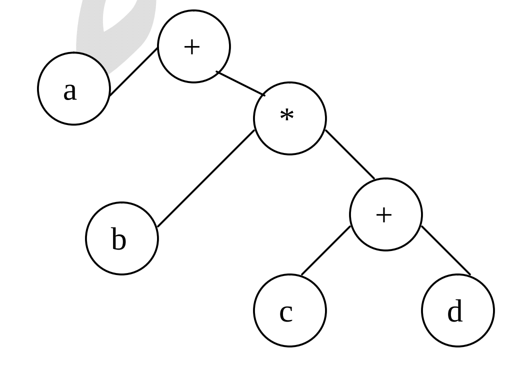
أ-S^ (a+b^n)



ب- X: = 2 * (a-b / c)

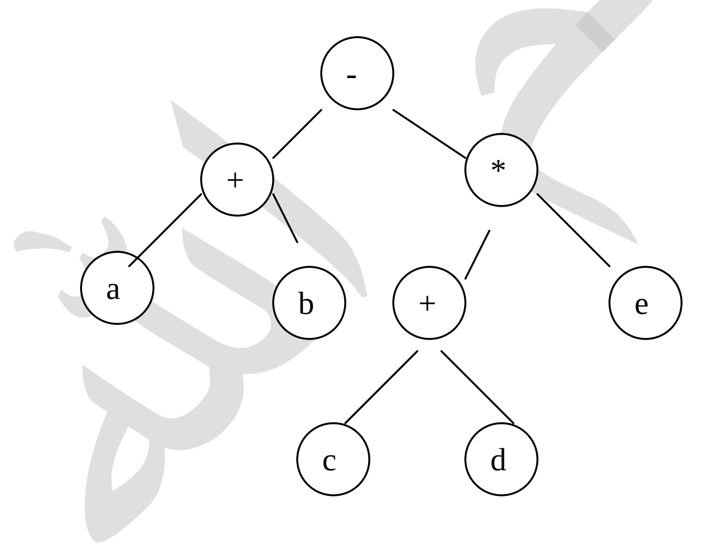
:=
X
*
2
_
a
/
b
c

ج- a+b*(c+d)



د- a^5+8*b^3-2*c-5

هـ- a+b-[(c+d) *e]



تمرين :

1 – ارسم الشجرة الثنائية التي تمثل التعبير الحسابي التالي :-

M: = L^ (2-b) - T^(3*b*5) /4P

2- ماهو عدد العقد المتفرعة فيها ...

3- ماهو عدد العقد التي درجتها (0)

4- اقطع من اسفل الشجرة لتحصل على شجرة ثنائية ارتفاعها (3) .

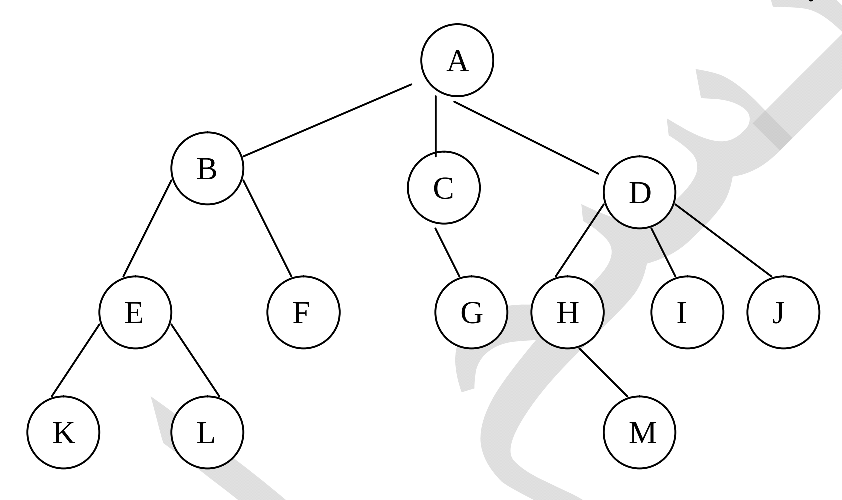
5- استعراض الشجرة الثنائية المتكونة بموجب الفقرة (4) بطريقتي الترتيب السابق والترتيب اللاحق .

6 – استخدم المصفوفة لتمثيل الشجرة المتكونة بموجب الفقرة (4) اعلاه .

هيكل الشجرة والمفاهيم الرياضية Tree Structure and Mathematical concepts

يمكن استخدام هيكل الشجرة لمثيل بعض المفاهيم الرياضية وبالعكس يمكن التعبير عن هيكل الشجرة بمفاهيم رياضية ..

لناخذ الشجرة التالية :



ان هذه الشجرة يمكن التعبير عنها رياضيا باستخدام مخططات فن (venn diagrams) كالاتي :

A

B C D

E F H

G I J

M

K L

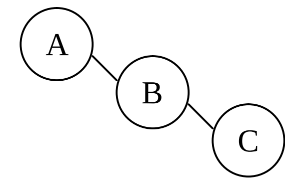
- ويمكن استخدام الاقواس المتداخلة (nested parenthesis)للتعبير عنها(A(B(E(K,L),F),C(G),D(H(M),I,J))

تمرين :ماهي صفات الاشجار الثنائية التي تعطي نفس المخرجات عنداستعراضها بكل من طريقتي

:

_الترتيب السابق (preorder) والترتيب البيني (inorder)

الجواب :هي الاشجار التي لاتحوي اي من عقدها على فرع ايسر اي ان Left link=nil كما في الشكل



_الترتيب الاحق (postorder) والترتيب البيني (inorder) الجواب :هي الاشجار التي لاتحتوي اي

من عقدها على فرع ايمن اي ان Rlink=nil كما في الشكل

_الترتيب السابق (preorder) والترتيب اللاحق (postorder)

الجواب :- هي الاشجار التي تتكون من عقدة واحدة تمثل الجذر وليس لها فرع ايمن او فرع ايسر وهذا يعني من الناحية البرمجية Llink := nil , RLink :=nil

٦-٨ تحويل الشجرة الاعتيادية الى شجرة ثنائية Transformation of atree into a binary tree

لتحويل الشجرة الاعتيادية الى شجرة ثنائية تتبع خطوات الخوارزمية التالية:

١- جذر الشجرة الاعتيادية يصبح هو جذر الشجرة الثنائية

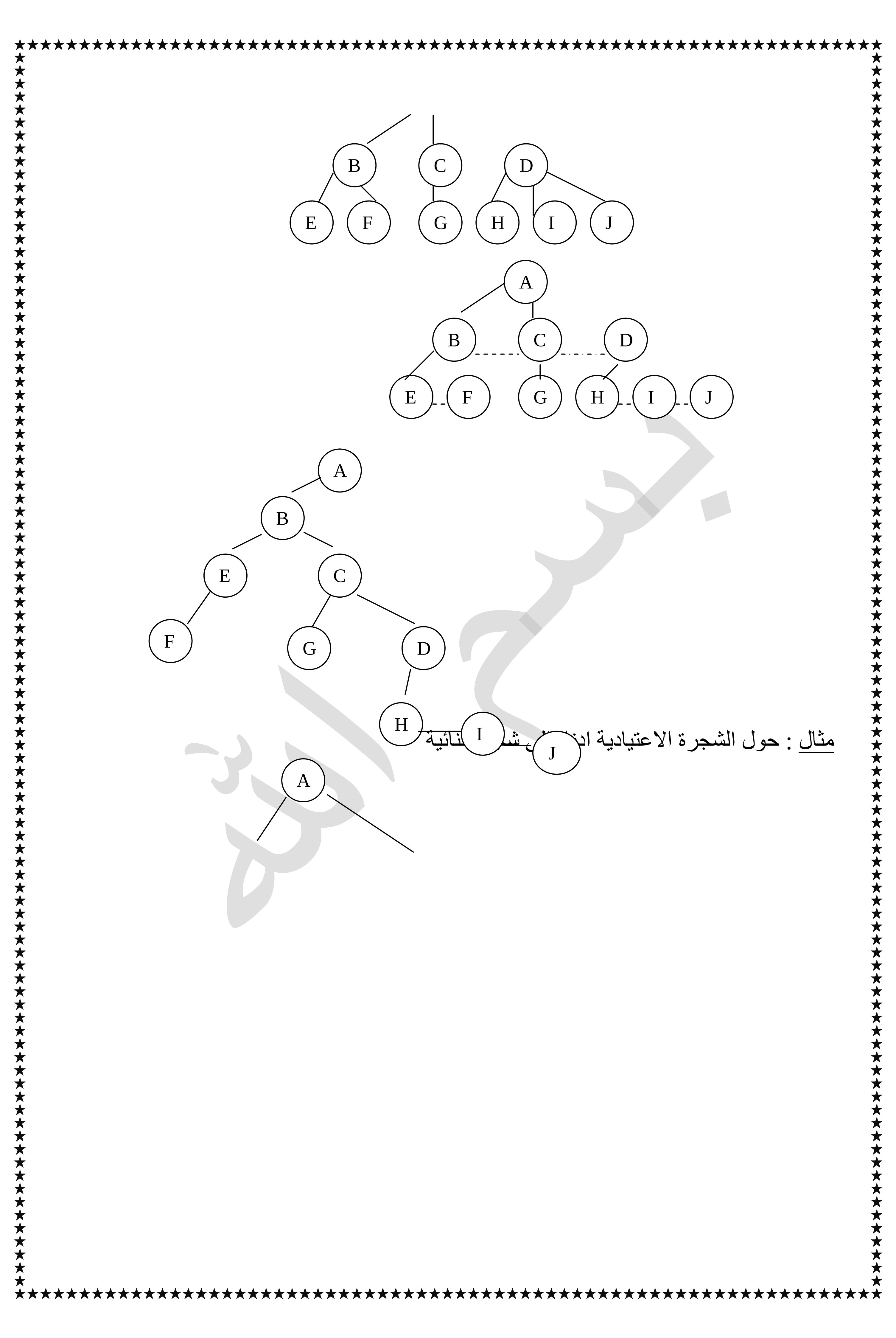
٢- الابن الايسر life child للشجرة الثنائية يكون نفسه الابن الايسر من الشجرة الاعتيادية

٣- ان اخوة (brothers) هذا الابن لايسر في الشجرة الاصلية (الاعتيادية) يصبحون الفرع (الابن) الايمن له في الشجرة الثنائية

٤- نعيد نفس الخطوات واعتبار الابن الايسر هو الجذر

٥-

مثال: لناخذ الشجرة الاعتيادية ادناه ونحولها الى شجرة ثنائية

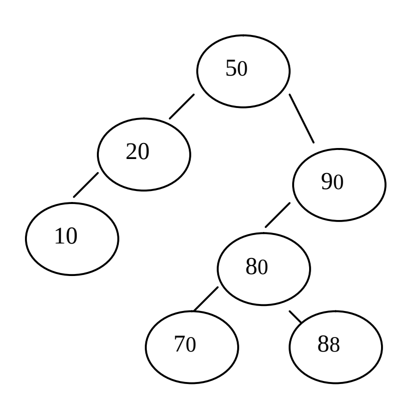


مثال : حول الشجرة الاعتيادية ادناه الى شجرة ثنائية

9-6 شجرة البحث الثنائية Binary Search Tree

هي شجرة ثنائية مرتبة تكون قيمة عنصر الفرع (الابن) الايسر لأية عقدة فيها هو اقل من قيمة عنصر تلك العقدة باعتبارها الاب وتكون قيمة عنصر الفرع (الابن) الايمن اكبر من قيمة عنصر العقدة (الاب)

<u>او بعبارة اخرى :</u> هي شجرة ثنائية مرتبة (Binary ordered type) تكون القيمة البيانية لاية عقدة فيها هي اكبر من القيمة البيانية للفرع الايسر واصغر من القيمة البيانية للفرع الايمن كما في الأشكال الاتية :



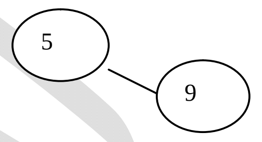
تمرين : ارسم شجرة البحث الثنائي للعناصر التالية:

20,4,6,12,8,3,7,9,5

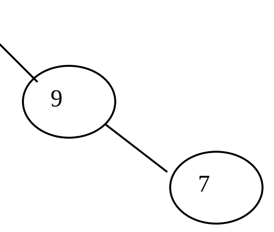
الجواب :

١-لناخذ العنصر الاول (٥) ونعتبر العقدة الجذر (٥)

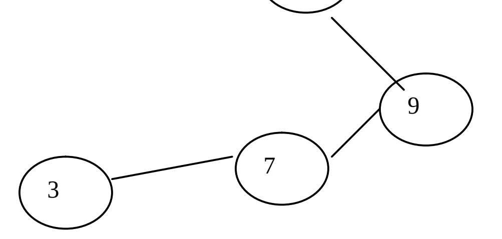
٢-خذ العنصر الثاني (٩) ولكونه اكبر من الجذر (٥) فيكون هو فرعا ايمنا للجذر



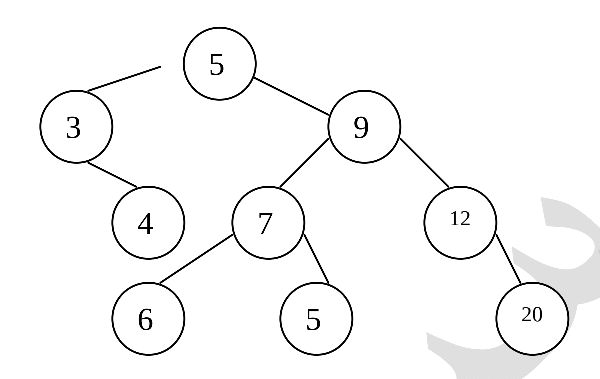
٣- ناخذ العنصر التالي (٧) وهو اكبر من عنصر الجذر (٥) فنذهب للفرع الايمن ونجده اصغر من (٩) فيكون فرعا ايسر للعقدة (٩)



٤- ناخذ العنصر (٣) وهو اصغر من عنصرالجذر (٥) فنضعه في يساره

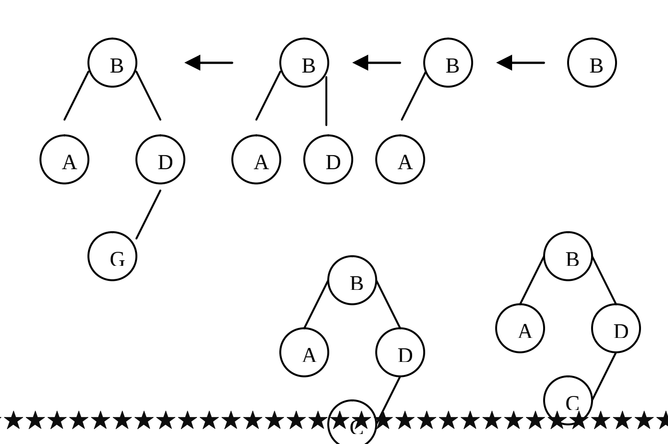


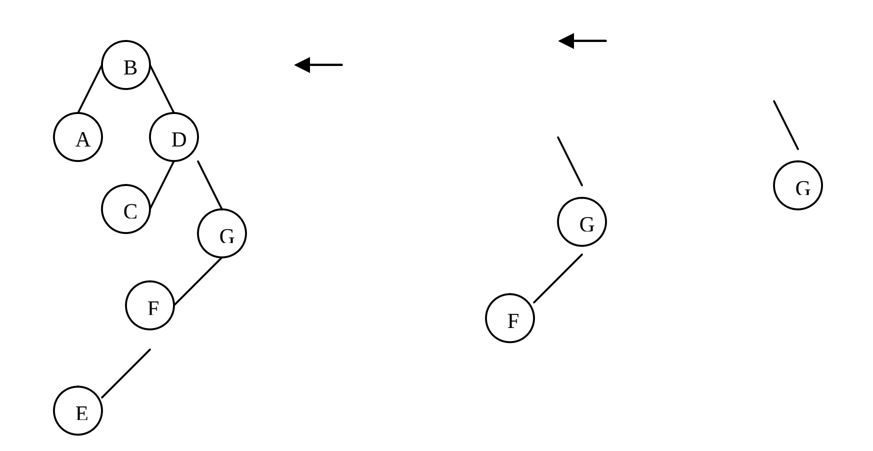
٥- نستمر بهذه الطريقة باخذ العنصر الجديد ومقارنته مع عناصر الشجرة ابتداء من الجذر وسنحصل على الشجرة بشكلها النهائي



تمرين : ارسم شجرة البحث الثنائية للعناصر التالية ;G ,E,C,A,F,B,D

تمرين : ارسم شجرة البحث الثنائية للعناصر التالية : E,F,G,C,D,A,B





تمرين : ارسم شجرة البحث الثنائية لعناصر التالية : G ,F,E,D,C,B,A

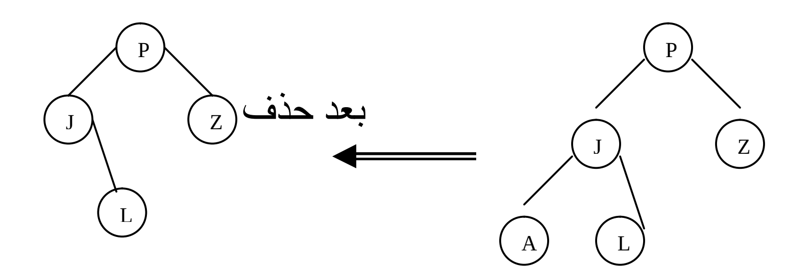
A
B
C
D
E
F
G

خطوات حذف عنصر من شجرة البحث الثنائية

ا- حذف عقدة نهائية (ورقة)

1 – نأخذ العقدة ونجعل قيمة مؤشر عقدة الاب اليها (nil)

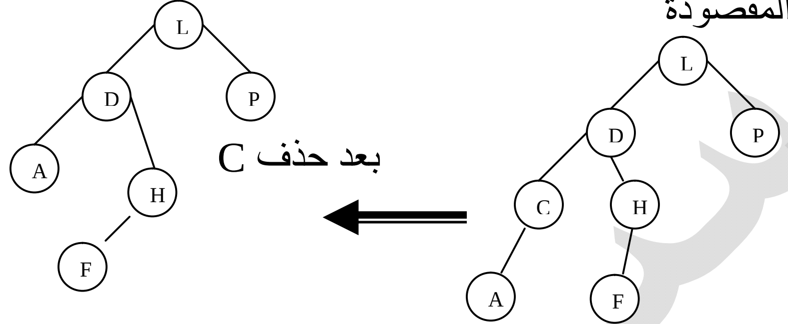
2– نلغي (free) العقدة



<u>ب- حذف عقدة لها فرع (ابن) واحد</u>

1-نجعل مؤشر اب العقدة (node father) يشير الى العقدة الابن

2 – نلغي (free) العقدة المقصودة



<u>ج - حذف عقدة لها فرعان</u>

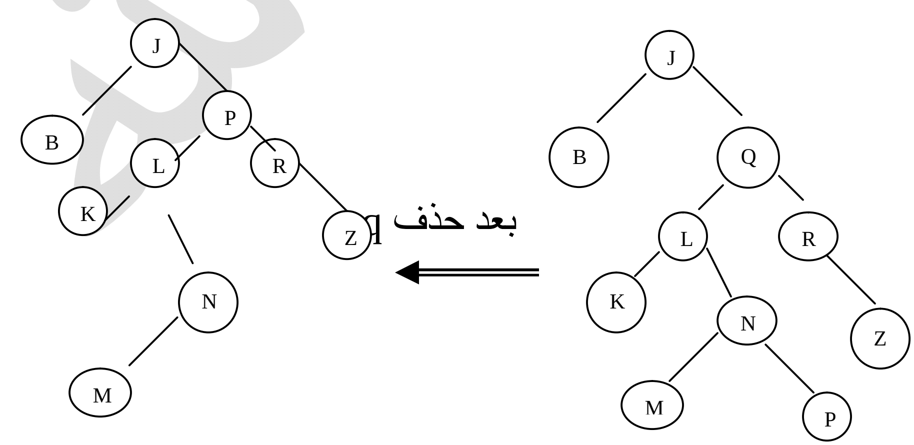
1 – تستبدل العقدة المطلوب حذفها بالعقدة التالية لها بالقيمة وهذه تستحصل من الشجرة الفرعية اليسرى او الشجرة الفرعية اليمنى بالنسبة للعقدة.

٢- ناخذ الشجرة الفرعيةاليسرى للعقدة (اي العقدة التي في يسار العقدة المطلوب حذفها).

+ اذا لم يكن لها فرع ايمن فانها تصبح البديل

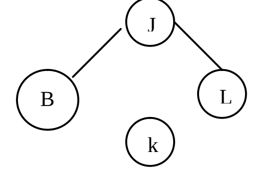
+اذا كان لها فرع ايمن فاننا ناخذ العقدة في اقصى اليمين لتصبح البديل.

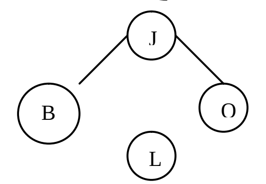
تمرين:



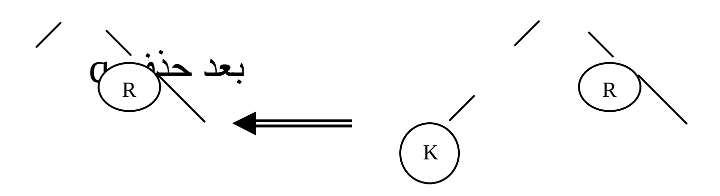
لحذف العقدة q التي لها فرعان R L نجد ان الشجرة الفرعية اليسرى L لها فرع ايمن ولهذا فناخذ اقصى اليمين L وهو P لتصبح بديل العقدة Q

تمرين





Z Z



لحذف العقدة q التي لها فرعان L , R نجد ان الشجرة الفرعية اليسرى L ليس لها فرع ايمن ولهذا فانها تصبح البديل اي ان العقدة L تحل في مكان العقدة q

برنامج فرعي : بصيغة التكرار (iteration) لايجاد عقدة في شجرة البحث الثنائية (binary search tree).

```
void findnode(struct node*p,int value)
{
 int found=0;
 while((p!=NULL)&&(!found))
   {
    if(p->data==value)
    found=1;
    else
     {
      if(p->data>value)
      p=p->left;
      else
      p=p->right;
     }
    }
}
```

برنامج فرعي : بصيغة الاستدعاء الذاتي (recursion) شجرة البحث الثنائية(binary search tree).

```
void btsearch(struct node*p,int key)
{
 if(p!=NULL)
  {
   if(p->data==key)
   cout<<"the key is found"<<endl;
   else if(p->data<key)
   btsearch(p->right,key);
   else
   btsearch(p->left,key);
```

```
  }
 }
```

برنامج -١٣ : تمثيل الشجرة الثنائية المرتبة (binary search tree) وعمليات الاضافة والمسح (traversing).

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node{
            int data;
            struct node*Rlink,*Llink;

  }*t,*r,*p1,*x,*h,*root;
int l,f,d,m;
void create(struct node*r,struct node*p1)
{
 if(f==0)
 {
  struct node*p;
  p=new node;
  cin>>p->data;
  p->Rlink=NULL;
  p->Llink=NULL;
  f=1;
  p1=p;
  d=p->data;
 }
 if(r!=NULL)
 if(d>=r->data)
 {
  if(r->Rlink==NULL)
  r->Rlink=p1;
  else
  create(r->Rlink,p1);
 }
 else
 {
  if(r->Llink==NULL)
```

```
  r->Llink=p1;
  else
  create(r->Llink,p1);
 }
}
void preorder(struct node*root)
{
 if(root!=NULL)
  {
   cout<<root->data;
   preorder(root->Llink);
   preorder(root->Rlink);
  }
}
void postorder(struct node*root)
{
 if(root!=NULL)
  {
   postorder(root->Llink);
   postorder(root->Rlink);
   cout<<root->data;
  }
}
void inorder(struct node*root)
{
 if(root!=NULL)
  {
   inorder(root->Llink);
   cout<<root->data;
   inorder(root->Rlink);
   }
}
void insert(struct node*h)
{
 if((h->Rlink!=NULL)&&(x->data>h->data))
 {
  h=h->Rlink;
```

```
 insert(h);
 }
else if((h->Rlink==NULL)&&(x->data>h->data))
h->Rlink=x;
else if(h->Llink!=NULL)
 {
 h=h->Llink;
 insert(h);
 }
else if(x->data<h->data)
h->Llink=x;
}
void main()
{
 int i;
 clrscr();
 cout<<"input the no.of nodes"<<endl;
 scanf("%d",&m);
 r=new node;
 cout<<"input the data field of each node"<<endl;
 cin>>r->data;
 r->Rlink=NULL;
 r->Llink=NULL;
 t=r;
 p1=NULL;
 for(i=1;i<=m-1;i++)
  {
   f=0;
   r=t;
   create(r,p1);
  }
  r=t;
  cout<<"the output of the preorder traversing is:"<<endl;
  preorder(r);
  getch();
  clrscr();
  cout<<"the out put of the inorder traversing is:"<<endl;
```

```
 inorder(r);
 getch();
 clrscr();
 cout<<"the output of the postorder traversing is:"<<endl;
 postorder(r);
 getch();
 clrscr();
 cout<<"to insert new node"<<endl;
 cout<<"input the new value"<<endl;
 h=t;
 x=new node;
 cin>>x->data;
 x->Llink=NULL;
 x->Rlink=NULL;
 insert(h);
 cout<<"after insertion the inorder traversing is:"<<endl;
 r=t;
 inorder(r);
 getch();
}
```

اسئلة الفصل

1- draw the tree of (height=5 , no. of nods>=17 , and at least two nodes of degree3)

i- represent this tree using two ways

+ venn diagrams

+nested parenthesis

ii- traverse the tree in preorder and post order

iii- transfer the above tree into binary tree

iv- traverse the resulting binary tree using in order method

v- represent the resulting binary tree using array

2- draw the binary trees which represents the following arithmetic expressions:

i- R*T^2-M*(L-E/F)

ii- a/b+c/d^2*(e-f)

iii- A:=b*c+(3-d*e)/f^4

3- the following figure shows an array representation for a binary tree

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X	M	F	R	-	T	N	-	A	-	-	Y	-	-	W

4- write an algorithm to transfer the general tree int o binary tree.

5- Write a procedure to find a node (with certain value)in the binary search tree.
6- Explain how a dynamically allocated record can be used to represent the binary tree.
What are the advantages and disadvantages of this representation over the array representation?
7- draw the binary search tree for the following sequence of keys(80, 10, 44, 25, 30, 8, 96, 70, 90, 33, 100) And perform the following operations in sequence showing the tree after each operation:
i- insert 75, 20
ii- delete 44
iii- insert 72, 92
iv- delete 8, 100
8- put true or false for each of the following:
i- The terminal nodes of a binary tree occur in the same relative position in pre order, in order and post order traversing out put.
ii- If we are given the pre order and in order traversing output of a binary tree, we can construct the tree.
iii- If we are given the post order and in order traversing output of binary tree, we can construct yhe tree.
iv- If we are given the pre order and post order traversing output of binary tree, we can construct yhe tree.
v- The terminal nodes of a tree (not binary) occur in the same relative position in pre order & post order traversing output.

الفصل السابع

Sorting and searching *الترتيب والبحث*

٧-١ الترتيب (Sorting)

٧-١-١ الغرض من الترتيب

٧-١-٢ خطوات عملية الترتيب

٧-١-٣ انواع خوارزميات الترتيب

I- الترتيب الداخلي (internal Sort)

II- الترتيب الخارجي (External Sort)

٧-١-٤ العوامل الرئيسية المحددة لاختيار خوارزمية الترتيب

٧-١-٥ ترتيب الفقاعة (Bubble Sort)

٧-١ –٦ الترتيب بالاختيار (Selection Sort)

٧-١-٧ الترتيب بالاضافة (Insertion Sort)

٧-١-٨ الترتيب السريع (Quick Sort)

٧-١-٩ ترتيب الدمج (Merge Sort)

٧-٢ البحث (Searching)

٧-٢-١ انواع خوارزميات البحث

٧-٢-٢ البحث التسلسلي (Sequential)

٧-٢-٣ البحث الثنائي (Binary Search)

٧-١ الترتيب Sorting

هي عملية ترتيب مجموعة من العناصر البيانية وفق قيمة حقل (أو حقول) يسمى المفتاح (key) بصورة تصاعدية (ascending) أو بصورة تنازلية (descending) .

٧-١-١ الغرض من الترتيب

تتعدد اغراض عملية الترتيب واهمها :
- لزيادة كفاءة خوارزمية البحث عن عنصر ما.
- لتبسيط معالجة الملفات .
- لحل مشكلة تشابه القيود.

٧-١-٢ خطوات عملية الترتيب

تتلخص خطوات خوارزمية الترتيب بكافة انواعها بالمراحل التالية :
١- قراءة حقل المفتاح .
٢- ٢- الاستدلال (استنتاج)موقع العنصر في الترتيب الجديد.
٣- نقل العنصر البياني الى موقعه الجديد.

٧-١-٣ انواع خوارزميات الترتيب

أ-الترتيب الذي يحدث في الذاكرة الرئيسية للحاسوب (main memory) عندما يكون حجم البيانات مناسباً(ليس كبيراً) للخزن الذاكرة.
ومن أهم انواعه:
- الترتيب بالاختيار Selection Sort
- ترتيب الفقاعة Bubble Sort (exchange)
- ترتيب الإضافة Insertion Sort
- الترتيب السريع Quick Sort
- ترتيب الأساس Radix Sort
- الترتيب الكومي Heap Sort
- ترتيب شيل Shell Sort

ب -الترتيب الخارجي External Sort

وهو ترتيب البيانات المخزو نة في أوساط الخزن الثا نوية على شكل ملفا ت عند ما يكون حجم البيانات كبير جدا بحيث يتعذر استيعابها كلها في الذاكرة في وقت واحد إثناء عملية الترتيب
ومن أهم أنواعه
١ الترتيب بالدمج ذي المسارين Tow-way-Merge Sort
٢ الترتيب بالدمج متعدد المسارات K-way-Merge Sort
٣ الترتيب بالدمج المتوازن ذي المسارين Balanced Two- way -Merge
٤ الترتيب بالدمج متعدد الاطوار polyhase Tow -way -Merge

7-1-4 العوامل الرئيسية المحددة لاختيار خوارزمية الترتيب

ان اختبار أي من خوارزميات الترتيب يجب ان يكون في ضوء عدد من العوامل من أهمها:
1- حجم البيانات المخزونة .
2- نوع الخزن (الذاكرة الرئيسية ، قرص ، شريط).
3- درجة ترتيب البيانات (غير مرتبة ، شبه مرتبة).

7-1-5 ترتيب الفقاعة Bubble Sort

أي أن القيمة الصغيرة تطفو)أن فكرة هذه الطريقة تتضمن اختبار اصغر القيم ووضعها في القائمة وفيما يأتي خطوات الخوارزمية (السطح

1- في المرحلة الأولى:(first pass) :نقارن العنصرين في الموقعين (n),(n-1) ونبادل موقعها ليكون الاصغر قبل الاخر،ونستمر لاعلى القائمة لحين الوصول الى مقارنة العنصر في الموقع الثاني مع العنصرفي الموقع الاول.

2- في المرحلة الثانية (second pass): نقارن بنفس الطريقة السابقة ولكن من العنصر في الموقع (n)الى العنصر في الموقع الثاني
1-لان الموقع الاول اختير فيه العنصر الأقل قيمة في الخطوة السا بقة
3 - ذكر الخطوات اعلاه ل (n-1) من المراحل .
مثال:لنقيم بترتيب القائمة ٨، ٣، ٩ ،٧ ، ٢ تصاعديا

المرحلة الرابعة	المرحلة الثالثة	المرحلة الثانية	المرحلة الاولى	القائمة الاصلية
1	2 1	3 2 1	4 3 2 1	
2	2 2	2 2 2	8 8	8
3	3 3	3 8 8	2 3	3
7	7 8	8 3 3	3 2	9
8	8 7	7 7 9	9 9 2	7
9	9 9	9 9 9	7 7 7	2

لاحظ ان عدد العناصر في القائمة هو (n=5) وعدد المراحل (n-1=4) وعدد الخطوات في كل مرحلة يتناقص بمقدار واحد عن عدد خطوات المرحلة السابقة لها.
ملاحظات:
- معدل عدد المقارنات Average no. of comparison هو (n2/2) حيث (n) يمثل عدد عناصر القائمة.
- معدل عدد التبديلات average no. of exchanges هو (n^2/4) .
- الطريقة جيدة عندما تكون العناصر شبه مرتبة وعددها ليس كبيراً ولا تحتاج مساحة خزنية كبيرة وبسيطة.
- ان وقت التنفيذ يبلغ O(n^2)

برنامج فرعي لترتيب الفقاعة

```
#include<iostream.h>
#include<conio.h>
const n=10;
int ar[n];
void bubblesort(int ar[n])
{
 int i,j;
 int x;
 for(i=0;i<n;i++)
```

```
  {
   for(j=n-1;j>i;--j)
    {
    if(ar[j]<ar[j-1])
     {
      x=ar[j];
      ar[j]=ar[j-1];
      ar[j-1]=x;
      }
     }
   }
}
```

برنامج -١٤ تمثيل خوارزمية ترتيب الفقاعة bubble sort

```
#include<iostream.h>
#include<conio.h>
const size=20;
int line[size];
int i,m;
void bubblesort(int ar[size],int n)
{
 int i,j;
 int x;
 for(i=0;i<n;i++)
  {
   for(j=n-1;j>i;--j)
    {
    if(ar[j]<ar[j-1])
     {
      x=ar[j];
      ar[j]=ar[j-1];
      ar[j-1]=x;
      }
     }
   }
}
void main()
{
```

```
clrscr();
cout<<"representation of bubble sort algorithm"<<endl;
cout<<"_______________________________________________"<<endl;
cout<<"how many data items you like to enter"<<endl;
cin>>m;
for(i=0;i<m;i++)
 {
  cout<<"enter the item"<<"\t"<<i+1<<endl;
  cin>>line[i];
 }
 bubblesort(line,m);
 cout<<"the sorted data is :"<<endl;
 for(i=0;i<m;i++)
 cout<<"\t"<<line[i];
 getch();
}
```

٧-١-٦ الترتيب بالاختيار selection sort

وتتلخص خوارزمية هذا الترتيب بالخطوات الاتية:

٣- ايجاد اصغر عنصر في القائمة واستبداله من موقعه مع العنصر في الموقع الاول في القائمة.

٤- ايجاد اصغر عنصر في المتبقي من القائمة واستبداله من موقعه مع العنصر في الموقع الثاني في القائمة .

٥- نستمر في هذه العملية لحين الوصول الى نهاية القائمة.

مثال: رتب القائمة التالية تصاعديا (8 3 9 7 2 6 4)

6	5	4	3	2	1	القائمة الأصلية
2	2	2	2	2	2	8
3	3	3	3	3	3	3
4	4	4	4	9	9	9
6	6	6	7	7	7	7
7	7	8	8	8	8	2
8	8	7	6	6	6	6
9	9	9	9	4	4	4

*عدد عناصر القائمة n=7 ، عدد المراحل n-1=6 no. of passes

ملاحظات: - معدل عدد المقارنات average no. of comparisons هو n/2*(n-1)

- معدل عدد التبديلات average no. of exchanges هو (n-1)

برنامج فرعي للترتيب بالاختيار:

```
#include<iostream.h>
#include<conio.h>
const n=20;
int line[n], i,m;
void slctsort(int data[n],int s)
{
 int i,k,j,item,x,y;
 for(i=0;i<s-1;i++)
  {
   k=i;
   item=data[i];
   for(j=i+1;j<s;j++)
    {
     if(data[j]<item)
      {
       x=data[j];
       data[j]=item;
       item=x;
      }
    }
   y=item;
   item=data[k];
   data[k]=y;
  }
}
```

برنامج ١٥: تمثيل خوارزمية الترتيب بالاختيار selection sort

```
#include<iostream.h>
#include<conio.h>
const n=20;
int line[n], i,m;
void slctsort(int data[n],int s)
{
 int i,k,j,item,x,y;
 for(i=0;i<s-1;i++)
  {
   k=i;
```

```
   item=data[i];
   for(j=i+1;j<s;j++)
    {
     if(data[j]<item)
      {
       x=data[j];
       data[j]=item;
       item=x;
      }
    }
   y=item;
   item=data[k];
   data[k]=y;
  }
}
void main()
{
 clrscr();
 cout<<"representation of selection sort algorithm"<<endl;
 cout<<"_____________________________________________"<<endl;
 cout<<"how many data items you like to enter"<<endl;
 cin>>m;
 for(i=0;i<m;i++)
  {
   cout<<"enter the item"<<i+1<<endl;
   cin>>line[i];
  }
  slctsort(line,m);
  cout<<"the sorted data is :"<<endl;
  for(i=0;i<m;i++)
  cout<<"\t"<<line[i];
  getch();
}
```

٧-١-٧ الترتيب بالاضافة inserting sort

تتلخص خطوات هذه الخوارزمية بما ياتي:

٦- نبدا بالعنصر الثاني i=2 في القائمة الاصلية ونقارنه مع العنصر الاول i=1 ونضعهم حسب الترتيب وليكن تصاعديا في مقدمة القائمة.

٧- ناخذ العنصر الثالث i=3 في القائمة الاصلية ونقارنه مع مقدمة القائمة التي تحوي العنصر الاول والثاني ونضعه في موقعه الصحيح معهم .

٨- ناخذ العنصر الرابع i=4 في القائمة الاصلية ونقارنه مع مقدمة القائمة التي تحتوي العناصر الثلاثة ونضعه في موقعه الصحيح بينهم.

٩- نستمر في هذه العملية لغاية العنصر الاخير وسنحصل على القائمة مرتبة

<u>مثال</u> : نرتب عناصر القائمة التالية تصاعديا (8 3 9 7 2 6 4)

6	5	4	3	2	1	القائمة الاصلية
2	2	2	3	3←	—3	8←
3	3	3	7	8	8←	3
4	6	7←	—8←	—9	9	←—9
6	7	8	9	7	7	7
7	8	9	2	2	2	2
8	9	6	6	6	6	6
9	4	4	4	4	4	4

*عدد العناصر n=7 ،*عدد المراحل n-1=6 ، اشارة السهم (←) توضح العنصر الذي اضيف

ملاحظات:- معدل عدد المقارنات comparisons هو n^2/4 حيث n يمثل عدد عناصر القائمة

- معدل عدد التبديلات n^4/4 average no. of exchanges

<u>برنامج فرعي للترتيب بالاضافة</u>

```
#include<iostream.h>
#include<conio.h>
const size=20;
int line[size],int i,m;
void insertionsort(int data[size],int n)
{
 int i,j,item;
 i=1;
 while(i<n)
   {
    j=i;
    while((j>=1) && (data[j]<data[j-1]))
      {
       item= data[j];
       data[j]=data[j-1];
```

```
        data[j-1]=item;
        j--;
       }
      i++;
     }
}
```

برنامج ١٦- تمثيل خوارزمية الترتيب بالاضافة inserting sort

```
#include<iostream.h>
#include<conio.h>
const size=20;
int line[size];
int i,m;
void insertionsort(int data[size],int n)
{
 int i,j,item;
 i=1;
 while(i<n)
   {
    j=i;
    while((j>=1) && (data[j]<data[j-1]))
       {
        item= data[j];
        data[j]=data[j-1];
        data[j-1]=item;
        j--;
       }
      i++;
     }
}
void main()
{
 clrscr();
 cout<<"representation of insertion sort algorithm"<<endl;
 cout<<"______________________________________________"<<endl;
 cout<<"how many data items you like to enter"<<endl;
 cin>>m;
 for(i=0;i<m;i++)
```

```
  {
   cout<<"enter the item"<<"\t"<<,i+1;
   cin>>line[i];
  }
  insertionsort(line,m);
  cout<<the sorted data is :"<<endl;
  for(i=0;i<m;i++)
  cout<<"\t"<<line[i];
  getch();
}
```

٧-١-٨ الترتيب السريع quick sort

ان خوارزمية هذا الترتيب تعتمد فكرة التجزئة واللصق، ففي ترتيب الفقاعة يتم مقارنة وتبديل العناصر المتجاورة لذلك اذا كان العنصر بعيد عن موقعه الصحيح ففي هذه الحالة سنحتاج الى عدد كبير من المقارنات.

ان خوارزمية الترتيب السريع تعالج هذا الضعف وتسمح باجراء المقارنات بين العناصر في المواقع المتباعدة وباقل عدد من المقارنات، اذ تعتمد فكرة التجزئة واللصق وتتلخص خطوات هذه الخوارزمية بالاتي:

١- اختيار احد عناصر القائمة في الوسط تقريبا وليكن x اي تقسم القائمة الى جزاين.

٢- يبدا المسح من الاتجاهين ، اي نبحث في النصف الاول (اليسار) من القائمة عن العنصر الذي قيمته اكبر من x ونبحث في النصف الثاني (اليمين) من القائمة عن العنصر الذي قيمته اصغر من x . نستبدل هذين العنصرين وذلك بجعل النصف الاول من القائمة يحتوي على عناصر اكبر من x .

٣- ناخذ النصف الاول من القائمة ونعالجه بنفس الاسلوب السابق (اي التجزئة واللصق) وهكذا مع النصف الثاني اي نستمر بالتجزئة واللصق تباعا لحين ترتيب جميع عناصر القائمة الكلية.

البرنامج الفرعي الاول للترتيب السريع

```
#include<iostream.h>
#include<conio.h>
const size=10;
int ar[size];
void swap(int *x,int *y)
{
 int temp;
 temp=*x;
 *x=*y;
```

```
 *y=temp;
}
void quicksort(int list[size],int f,int l)
{
 int i,j,x;
 i=f;
 j=l;
 x=list[(i+j)/2];
 do
  {
   while(list[i]<x)
   i++;
   while(x<list[j])
   j--;
   if(i<=j)
    {
     swap(&list[i],&list[j]);
     i++;
     j--;
    }
   }while(i<=j);
   if(f<j)
   quicksort(list,f,j);
   if(i<l)
   quicksort(list,i,l);
}
```

مثال : استخدم الخوارزمية quick sort للترتيب السريع (باعتماد العنصر الوسط محورا للترتيب) لترتيب مجموعة القيم التالية تصاعديا.(20, 85, 60, 75, 70, 88, 50, 90, 33, 95)

أ- نفترض ان العناصر مخزونة في المصفوفة list وبالصورة الاتية :

F=1, l=10, x= list (5)=70, i=1, j=10

1	2	3	4	5	6	7	8	9	10
20	85	60	75	**70**	88	50	90	33	95
I									j

ب- I=2, j=9, list(2) <x,x< list (9), i=j

1	2	3	4	5	6	7	8	9	10
20	85	60	75	**70**	88	50	90	33	95
	I=2							j=9	

ج-

1	2	3	4	5	6	7	8	9	10
20	33	60	75	**70**	88	50	90	85	95
		I=3					j=8		

د- I=4,j=7, list (4)< x , x < list (7), i<=j

1	2	3	4	5	6	7	8	9	10
20	33	60	75	**70**	88	50	90	85	95
			I=4			j=7			

هـ-

1	2	3	4	5	6	7	8	9	10
20	33	60	50	**70**	88	75	90	85	95
				I=5	j=6				

و- I=5, j=5, list (5) <x, x< list (j)

1	2	3	4	5	6	7	8	9	10
20	33	60	50	**70**	88	75	90	85	95
				I=j=5					

عند هذه الخطوة جزئت القائمة الى قسمين القسم الايسر يحتوي على جميع الاعداد التي قيمتها اقل من 70 والقسم الايمن يحتوي على جميع الاعداد التي قيمتها اكبر من 70 .
اما الخطوة التالية فهي تنفيذ الخوارزمية بصورة متكررة على كل جزء بنفس الطريقة اي استدعاء (1,4) quick sort فيما يتعلق بالجزء الذي عناصره في المواقع ١الى ٤ واستدعاء (6,10) quick sort فيما يتعلق بالجزء الذي عناصره في الموقع من ٦-١٠ وهكذا تستمر عملية تكرار التجزئة والترتيب.

طريقة اخرى : في الخطوات السابقة نلاحظ اختيار العنصر الواقع وسط القائمة ليكون محور (مركز) المقارنة ليكون (pivot) ونقارن معه العناصر الاخرى ، وهذه طريقة اخرى تتضمن اختيار العنصر في الموقع الاول لهذا البغرض وبموجب خطوات الخوارزمية الاتية:

١- اختيار العنصر في الموقع الاول ليكون محور pivot التجزئة.
٢- نقل هذا العنصر واخلاء موقعه.
٣- نبدا مسح العناصر من الجهة الاخرى اي اليمين ونقارن كل عنصر مع عنصر المحور pivot value
٤- عند ايجاد عنصر اصغر من العنصر المحور ينقل ذلك العنصر الى الموقع الذي كان فيه العنصر المحور ويبقى موقعه خاليا.
٥- لمسح العناصر من جهة اليسار باتجاه اليمين ونقارن هذه العناصر مع العنصر المحور فاذا وجدنا عنصرا اكبر منه ننقله الى الموقع الخالي ويترك موقعه خاليا.
٦- ننتقل الى جهة اليسار لنمسح العناصر باتجاه اليمين لحين الوصول الى عنصر اكبر نمن العنصر المحور وننقله الى الجهة الاخرى وبنفس الاسلوب ننتقل الى الجهة اليمنى.

٧- بعد توزيع العناصر التي اكبر من العنصر المحور في اليمين والعناصر التي اصغر منه في اليسار نعيد العنصر المحور الى الموقع الخالي ليصبح هو الفاصل بينهما.
٨- نكرر الخطوات السابقة على عناصر القائمة عدا العنصر الاول ثم نكرر مرة اخرى على عناصر القائمة عداد العنصرين الاولين وهكذا لحين انتهاء عملية الترتيب.

البرنامج الفرعي الثاني للترتيب السريع

```
#include<iostream.h>
#include<conio.h>
const size=10;
int ar[size];
void quicksort2(int list[size],int f,int l)
{
 int i,j,x;
 while(l>f)
   {
    i=f;
    j=l;
    x=list[f];
    while(i<j)
      {
       while(list[j]>x)
       j--;
       list[i]=list[j];
       while((i<j)&&(list[i]<=x))
       i++;
       list[j]=list[i];
      }
    list[i]=x;
    quicksort2(list,f,i-1);
    f=i+1;
   }
}
```

مثال : استخدم خوارزمية الترتيب السريع (باعتماد العنصر الوسط محورا للترتيب) لترتيب مجموعة القيم التالية تصاعديا. (95 ,40 ,70 ,30 ,90 ,38 ,75 ,10 ,85 ,50)

أ- نفترض ان العناصر مخزونة في المصفوفة بالصورة الاتية: F=1, l=10, i=1, j=10, x= list(1)=50

1 2 3 4 5 6 7 8 9 10

50 85 10 75 38 90 30 70 40 95
I=1 J=10

ب- i=1, j=9, list(9)> x

1 2 3 4 5 6 7 8 9 10
50 85 10 75 38 90 30 70 **40** 95
I=1 (move it)j

ج- i=1, j=9, list (i)= list (j)

1 2 3 4 5 6 7 8 9 10
40 85 10 75 38 90 30 70 40 95
I j

د- i=2, j=9, list (i) <=x

1 2 3 4 5 6 7 8 9 10
40 **85** 10 75 38 90 30 70 **40** 95
Move it j

هـ- i=2, j=9, list (j)= list(i)

1 2 3 4 5 6 7 8 9 10
40 **85** 10 75 38 90 30 70 **85** 95
I j

و- i=2, j=7, list (j)> x

1 2 3 4 5 6 7 8 9 10
40 **85** 10 75 38 90 **30** 70 85 95
I move it j

ز- i=2, j=7, list(i)=list(j)

1 2 3 4 5 6 7 8 9 10
40 **30** 10 75 38 90 **30** 70 85 95
I j

ح- i=4, j=7, list(i)<=x

1 2 3 4 5 6 7 8 9 10
40 30 10 **75** 38 90 **30** 70 85 95
I move it j

ط- i=4, j=7, list (j)=list(i)

1 2 3 4 5 6 7 8 9 10
40 30 10 **75** 38 90 **75** 70 85 95
I j

ك- i=4, j=5 list (j)>x

1 2 3 4 5 6 7 8 9 10
40 30 10 **75** **38** 90 75 70 85 95

I j

ل- i=4, j=5, list (i)= list (j)

1	2	3	4	5	6	7	8	9	10
40	30	10	**38**	**38**	90	75	70	85	95

I j

م- i=5, j=5

1	2	3	4	5	6	7	8	9	10
40	30	10	**38**	**38**	90	75	70	85	95

I j

ن- i=5, j=5, i<j, list(i)=x

1	2	3	4	5	6	7	8	9	10
40	30	10	38	50	90	75	70	85	95

I j

عند هذه الخطوة جزئت القائمة الى قسمين ، القسم الايسر يحتوي على جميع الاعداد التي قيمتها اقل من ٥٠ والقسم الايمن يحتوي على جميع الاعداد التي قيمتها اكبر من ٥٠ . في الخطوة التالية تنفيذ نفس الخوارزمية على الجزء الايسر اي استدعاء quick stor2(f,i-1) ثم على الجزء الايمن لحين ترتيب جميعغ عناصر القائمة .ان تحليل خوارزمية الترتيب السريع يشير الى معدل عدد المقارنات (comparisons) هو $(n \log^2 n)$ومعدل عدد التبادلات (exchanges) هو $(1/2\ n \log^2 n)$ وان استخدام هذه الخوارزمية لترتيب عناصر قائمة مرتبة سيتطلب وقت تنفيذ طويل يتناسب مع (n2) بدلا من المعدل $(n \log^2 n)$.

برنامج -١٧ : تمثيل الخوارزمية الاولى للترتيب السريع باستخدام صيغة الاستدعاء الذاتي(recursion).

```
#include<iostream.h>
#include<conio.h>
const size=20;
int data[size],value;
int first,last,m,k;
void quicksort(int list[size],int lower,int upper)
{
 int i,j,x,item;
 i=lower;
 j=upper;
 item=list[(lower+upper)/2];
 do
  {
   while(list[i]<item)
   i++;
```

```
    while(item<list[j])
    j--;
    if(i<=j)
     {
      x=list[i];
      list[i]=list[j];
      list[j]=x;
      i++;  j--;
     }
    }while(i<=j);
    if(lower<j)
    quicksort(list,lower,j);
    if(i<upper)
    quicksort(list,i,upper);
 }
void main()
{
 int i;
 clrscr();
 cout<<"representation of quick sort algorithm"<<endl;
 cout<<"____________________________________________________"<<endl;
 cout<<"how many data items you like to enter"<<endl;
 cin>>m;
 for(i=0;i<m;i++)
  {
   cout<<"enter the item\t"<<i+1<<endl;
   cin>>data[i];
  }
  quicksort(data,0,m-1);
  cout<<<"the sorted data is :"<<<endl;
  for(i=0;i<m;i++)
  cout<<data[i];
  getch();
}
```

برنامج -١٨: تمثيل الخوارزمية الثانية للترتيب السريع (quick sort2) باستخدام صيغة الاستدعاء الذاتي

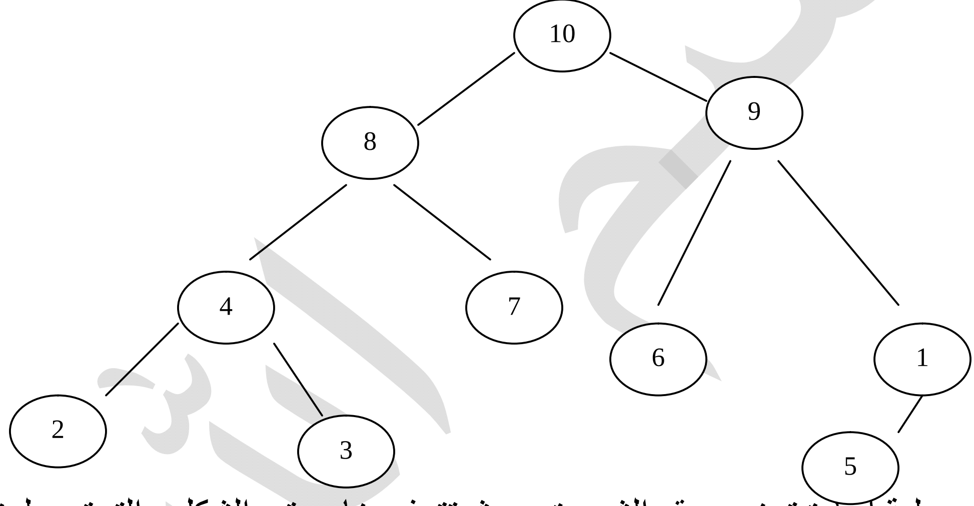
```
#include<iostream.h>
```

```
#include<conio.h>
const size=20;
int data[size],first,last,m,k;
void quicksort2(int list[size],int lower,int upper)
{
 int i,j,item;
 while(lower<upper)
   {
    i=lower;
    j=upper;
    item=list[lower];
    while(i<j)
      {
       while(list[j]>item)
       j--;
       list[i]=list[j];
       while((i<j)&&(list[i]<=item))
       i++;
       list[j]=list[i];
      }
    list[i]=item;
    quicksort2(list,lower,i-1);
    lower=i+1;
   }
}
void main()
{
 int i;
 clrscr();
 cout<<"representation of quick sort algorithm"<<endl;
 cout<<"__________________________________________________"<<endl;
 cout<<"how many data items you like to enter"<<endl;
 cin>>m;
 for(i=0;i<m;i++)
  {
   cout<<"enter the item\t"<<i+1;
   cin>>data[i];
```

```
 }
 quicksort2(data,0,m-1);
 cout<<"the sorted data is :"<<endl;
 for(i=0;i<m;i++)
 cout<<"\t"<<data[i];
 getch();
}
```

٧-١-٩ الترتيب الكومي heap sort:يعتمد هذا الترتيب فكرة بناء الاشجار الثنائية وتمثيلها في مصفوفة كما شرحنا في الفصل السادس . ولغرض وصف اسلوب خوارزمية هذا الترتيب نحتاج الى توضيح بعض المفاهيم.

الكومة heap : هي صيغة بيانية تتوفر فيها خاصتين،الاولى تتعلق بالشكل(shape)الذي يجب ان يكون شجرة ثنائية كاملة (compete binary tree)والخاصية الثانية تتعلق بترتيب العناصر ويعني ان قيمة كل عقدة يجب ان تكون اكبر او تساوي قيمة كل من عقدتيها الفرعيتين (طفليها) اليسرى واليمنى كما في الشكل التالي:



Reheap : هي عملية اعادة توزيع عقد الشجرة بحيث تتوفر خاصيتي الشكل والترتيب لعناصرها ابتداءا من عقدة الجذر والى ادنى عقدة في ادنى مستوى من خلال استبدال عقد الابناء التي قيمتها اكبر من عقدة الجذر لتصبح عقدة الجذر هي الاكبر.

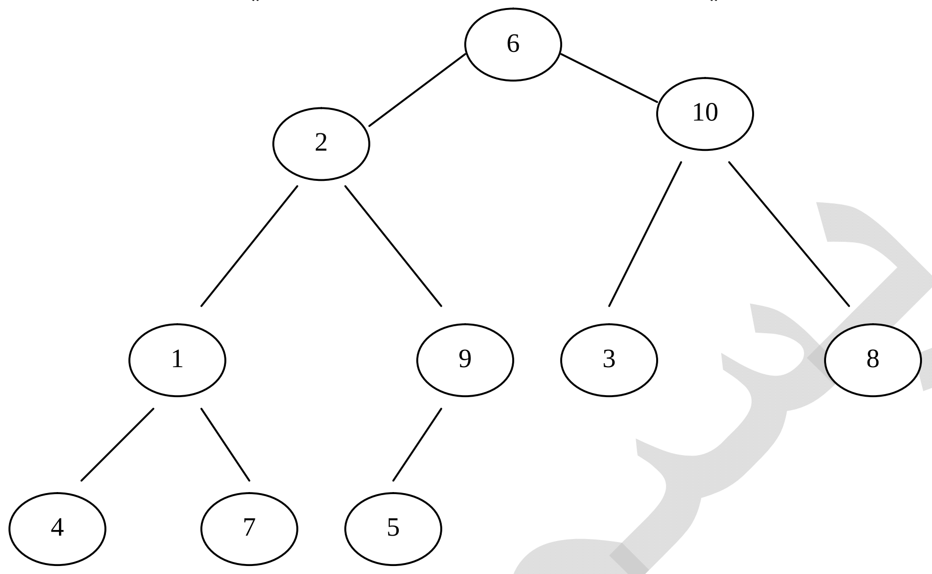
وصف خوارزمية الترتيب الكومي

في ترتيب الاختيار يكون البحث عن اصغر في مجموعة القيم ونقله الى موقعه الصحيح ،ويستمر البحث في المتبقي من عناصر القائمة عن اصغر عنصر فيها ليوضع في موقعه الصحيح بعد موقع العنصر السابق ، وهكذا الى ان يتم ترتيب القائمة . اما في هذا الترتيب (heap sort) فيمكن تلخيص خواته بما ياتي:

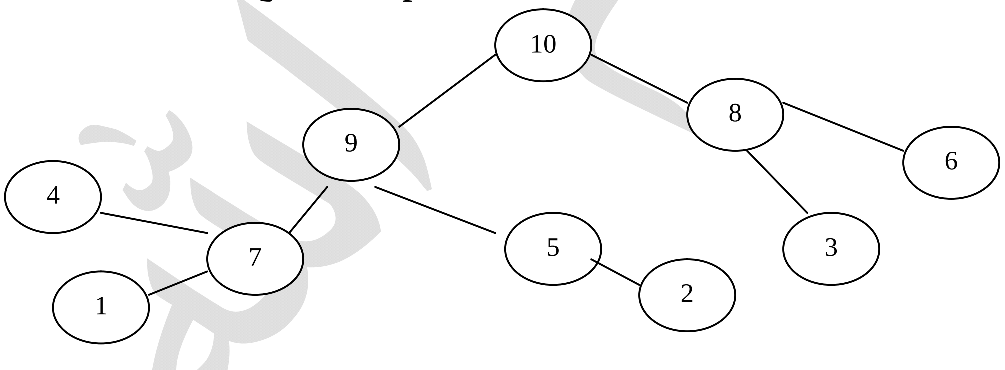
١- تستخدم المصفوفة لخزن البيانات الاولية غير المرتبة ، ثم تعتمد فكرة الكومة heap في ترتيب العناصر ثنائية الجذرها هو اكبر العناصر ويكون في الموقع الاول للمصفوفة ، اما فرعي الجذر (الايسر والايمن) فيكونا في الموقعين التاليين مباشرة.

٢- نأخذ الجذر من الكومة heap لانه يمثل اكبر قيمة ونضعه في موقعة الصحيح .

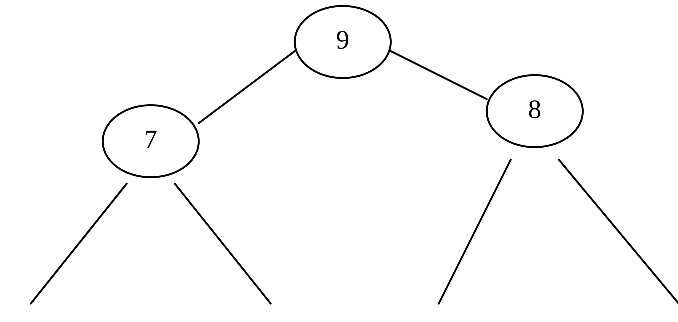
٣- اعادة ترتيب reheap العناصر المتبقية على شكل كومة جديدة heap اي شجرة ثنائية اخرى ، وهذا يعني ان العنصر الاكبر التالي سيكون هو الجذر فيها.
٤- نكرر الخطوات اعلاه لحين الحصول على القائمة المرتبة.
مثال : لناخذ المدخلات التالية : 6, 2, 10,, 1, 9, 3, 8, 4, 7, 5
الحل :
١- يمكن تمثيل هذه المدخلات في شجرة ثنائية وتخزن عناصرها في مصفوفة.

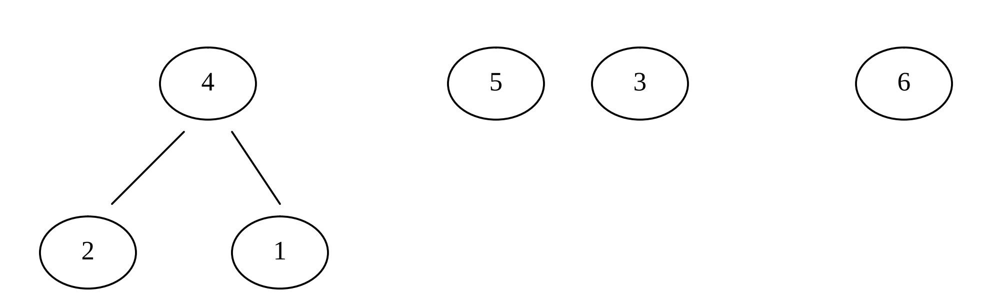


٢- عند اعادة ترتيب عناصر هذه الشجرة لتكوين الكومة heap تصبح :



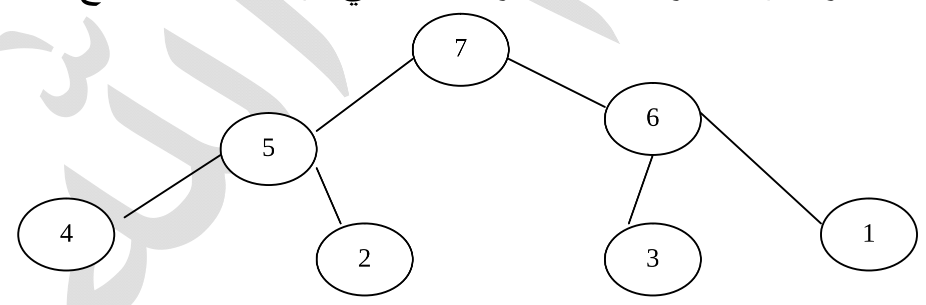
٣- تؤخذ قيمة الجذر 10 وهي اعلى قيمة ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصبح:



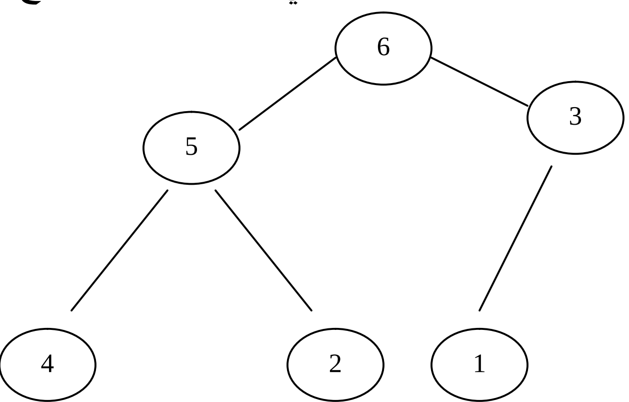


٤- تؤخذ قيمة الجذر 9 وتوضع بالترتيب بعد العنصر السابق 10 ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصبح:

٥- توخذ قيمة الجذر 8 ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصبح



٦- توخذ قيمة الجذر 7 ويعاد ترتيب العناصر التبقية في كومة جديدة لتصبح :



٧- تؤخذ قيمة الجذر 6 ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصبح:

٨- تؤخذ قيمة الجذر 5 ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصبح:

٩- تؤخذ قيمة الجذر 4 ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصبح:

١٠- تؤخذ قيمة الجذر 3 ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصبح :

١١- تؤخذ قيمة الجذر 2 ويعاد ترتيب العناصر المتبقية في كومة جديدة لتبقى فيها العقدة الاخيرة وقيمتها 1.

١٢- ان العناصر التي اخذت تباعا بعد تشكيل كل كومة وخزنت في المصفوفة التي اصبحت مرتبة كالاتي:

برنامج فرعي للترتيب الكومي:

```
#include<iostream.h>
#include<conio.h>
```

```
const n=10;
int ar[n];
void heapsort(int data[n],int n)
{
 int index;
 for(index=n/2;index<=0;index--)
 reheap(data,index,n);
 for(index=n-1;index<=1;index--)
  {
   swap(&data[0],&data[index]);
   reheap(data,0,index-1);
   display(data,n);
  }
}
```

ملاحظة:هذا البرنامج يستدعي برامج فرعية اخرى: display,swap,reheapموضحة في الصفحات التالية:

البرنامج الفرعي(display)

```
void display(int ss[n],int n)
{
 int i;
 for(i=0;i<n;i++)
 cout<<endl<<ss[i];
}
```

البرنامج الفرعي reheap

```
void reheap(int heap[n],int root,int bottom)
{
 int ok,max;
 ok=0;
 while((root*2<=bottom)&&(!ok))
   {
    if(root*2==bottom)
    max=root*2;
    else
    if(heap[root*2]>heap[root*2+1])
    max=root*2;
    else
    max=root*2+1;
```

```
    if(heap[root]<heap[max])
     {
      swap(&heap[root],&heap[max]);
      root=max;
     }
    else
    ok=1;
   }
}
```

البرنامج الفرعي(swap)

```
void swap(int *a,int *b)
{
  int c;
  c=*a;  *a=*b;  *b=c;
}
```

٧-١-١٠ ترتيب الدمج balanced two –way merge

هذه الطريقة هي من انواع الترتيب الخارجي وتتلخص الخوارزمية بالخطوات الاتية:

١- تقسيم القائمة (البيانات) الاصلية الى قائمتين متساويتين تقريبا ولتكن a,b

٢- نقارن العنصر الاول من القائمة a مع نظيره العنصر الاول من القائمة b ونضعهما بالترتيب في القائمة c .

٣- نقارن العنصر الثاني من القائمة a مع نظيره العنصر الثاني من القائمة b ونضعهما بالترتيب في القائمة d

٤- نكرر الخطوتين 2,3 وسنحصل على عناصر مزدوجة string of length 2 في كل من القائمتين c,d .

٥- بنفس الطريقة نقوم بدمج عناصر القائمتين c,d ونضعهما في القائمتين a,b وسنكون عناصرهما بطول 4.

٦- نعيد الطريقة بدمج عناصر القائمتين a,b ونضعهما في القائمتين c,d وستكون عناصرهما بطول 8 .

٧- نستمر بهذا الاسلوب لحين الحصول على القائمة النهائية المرتبة.

مثال:رتب تصاعديا هذه القائمة باستخدام خوارزمية الدمج(18, 23, 02, 50, 42, 63, 20, 28, 33, 47, 3)

الحل: ان عدد عناصر القائمة (n=1) وتجزئ الى قائمتين متساويتين بالعدد تقريبا.

A: 18, 23, 2, 50, 42
B: 63, 20, 28, 33, 47, 3
C: 18, 63, 2, 28, 42, 47
D: 20, 23, 33, 50, 3

A:18, 20, 23, 63, 3, 43, 47
B: 2, 28, 33, 50
C: 2, 18, 20, 23, 28, 50, 63
D: 3, 42, 47

A: 2,3,18,20,23,28,42,47,50,63

بعد سلسلة الخطوات اعلاه حصلنا على على عناصر القائمة مرتبة تصاعدية كما في a

٢-٧ البحث searching

هي عملية ايجاد عنصر معين في مجموعة من البيانات اذا كان ذلك العنصر موجود ، مثلا ايجاد اسم شخص في دليل الهاتف.ان عملية البحث قد تكون ايجابية عند وجود العنصر المطلوب وقد تكون سلبية في حالة كون العنصر غير موجود في قائمة البحث. وتكون عملية البحث فعالة عندما تكون العناصر البحث مرتبة وفق نسق (قيمة حقل) معين.

٧-٢-١ انواع خوارزميات البحث

١- البحث التسلسليsequential search
٢- البحث الثنائيbinary search
٣- البحث الكتلي block search

بحث الشجرة الثنائية binary tree search

٧-٢-٢ البحث التسلسليsequential search

وهي عملية البحث عن عنصر معين في قائمة من العناصر من خلال (استعراض) جميع عناصر القائمة من بدايتها وبالتسلسل لحين الوصول للعنصر المطلوب في حالة وجوده اوالوصول الى نهاية القائمة عند عدم وجوده لذا فان معدل عدد المقارنات سيكون (n/2) اي ان وقت تنفيذ هذه الخوارزمية سيكون O(n)

البرنامج الفرعي للبحث التسلسلي binary search

```
#include<iostream.h>
#include<conio.h>
const n=20;
int a[n];
void seqsearch(int n,int item)
{
 int i;
 i=n;
 while((i>=0)&&(item!=a[i]))
 i--;
 if(i>-1)
 cout<<"the item is found"<<endl;
```

```
 else
 cout<<"the item is not found"<<endl;
}
```

ان خوارزمية هذا البحث تفترض التفتيش عن عنصر معين في قائمة مرتبة (sorted) حسب تسلسل (ترتيب) معين ويمكن تلخيصها بالخطوات الاتية:

١- تحديد موقع العنصر الذي يقع في منتصف القائمة تقريبا.
٢- مقارنة العنصر المراد البحث عنه ليكن x مع النصر الذي يقع في منتصف القائمة.
٣- اذا كان العنصر المطلوب x مساويا للعنصر في الوسط ستنتهي عملية البحث هنا.
٤- اذا كان العنصر المطلوب x اقل من قيمة العنصر الذي يقع في المنتصف فان البحث سينحصر في الجزء الذي يضم القيم الاصغر وليكن الجزء الذي في القسم الايسر.
٥- اذا كان العنصر المطلوب x اكبر من قيمة العنصر الذي يقع في المنتصف فان البحث سينحصر في الجزء الذ يضم القيم الاكبر وليكن الجزء الذي يقع في القسم الايمن.
٦- في اي الحالتين (5,4) تتم معالجة ذلك الجزء بنفس الطريقة ، اي اختيار نقطة المنتصف والمقارنة لحين الوصول الى العنصر المطلوب.

في هذه الخوارزمية فان كل مقارنة ستؤدي الى تقليص عدد المقارنات اللاحقة الى النصف ولهذا فان اكبر عدد للمقارنات سيبلغ (log2n) عند البحث في قائمة عدد عناصرها n مع ملاحظة انه يجب ان تكون العناصر مخزونة في مصفوفة لانها ستكون في مواقع متعاقبة.

<u>البرنامج الفرعي للبحث الثنائي:</u>

```
#include<iostream.h>
#include<conio.h>
const n=20;
int a[n];
void binsearch(int a[n],int x,int n,int j)
{
 int upper,lower,mid;
 int found;
 lower=1;
 upper=n-1;
 found=0;
 while((lower<=upper)&&(!found))
   {
    mid=(lower+upper)/2;
    switch (compare(x,a[mid]))
     {
      case'>':lower=mid+1;break;
```

```
      case'<':upper=mid-1;break;
      case'=':
       {
        j=mid;
        found=1;
       }
      break;
     }
   }
}
```

ان البرنامج الفرعي أعلاه يستخدم الدالة compare المعرفة أدناه:

```
char compare(int x,int y)
{
 if(x>y)
 return('>');
 else
  {
   if(x<y)
   return('<');
   else return('=');
  }
}
```

<u>مثال</u> : في هذا المثال نوضح تنفيذ البرنامج الفرعي السابق bin search لايجاد القيمة المفتاحية 25 في قائمة عدد عناصرها n=8 .

a) Initialize: found= false,   key=25

1   2   3   4   5   6   7   8

first   last

b) iteration 1 : first<= last and not found

mid=(1+8)div2=4

list (4)<25 so move first to (mid+1)

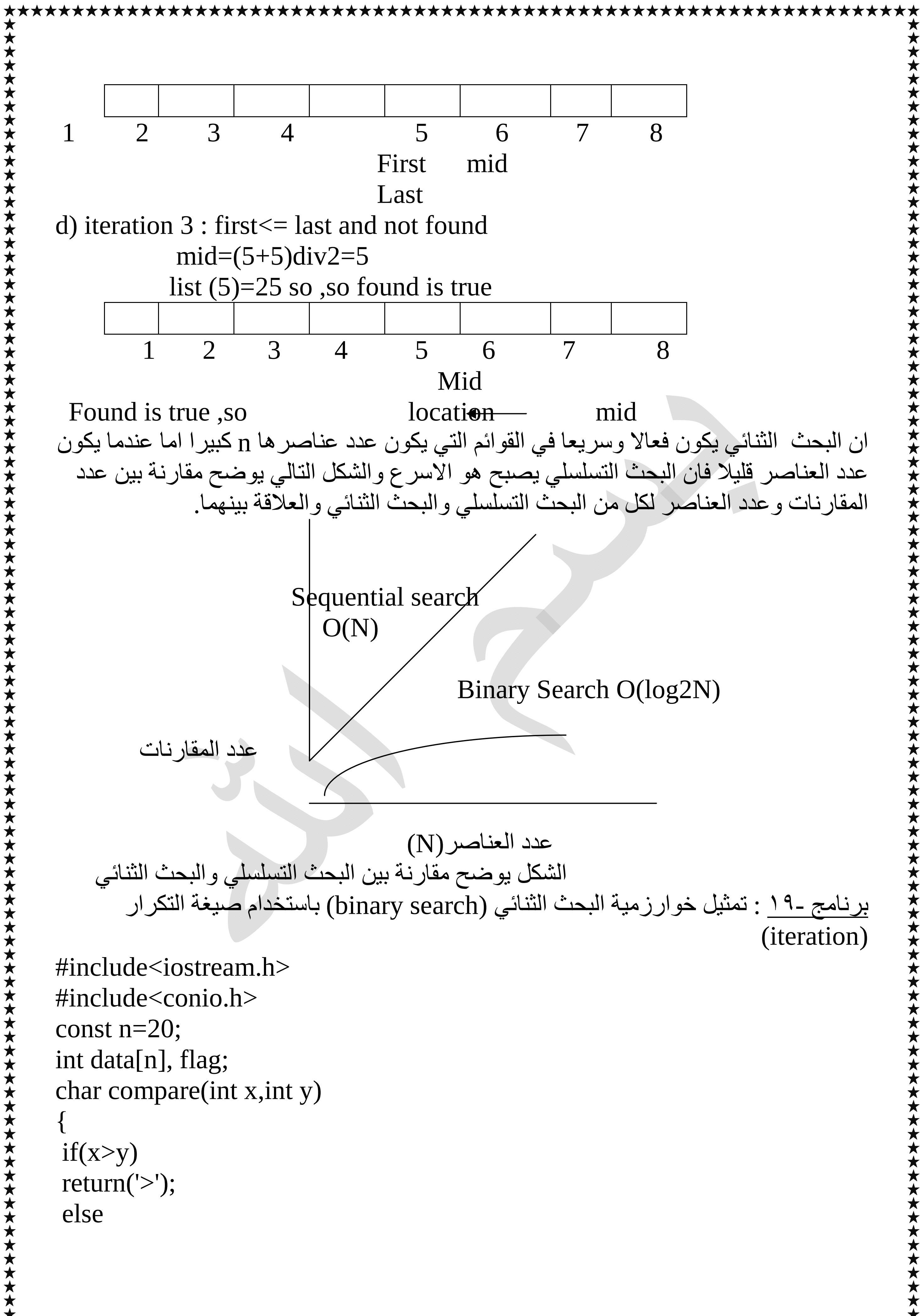
1   2   3   4   5   6   7   8

Mid   first   last

c)) iteration 2 : first<= last and not found

mid=(5+8)div2=6

list (6)<25 so move last to (mid-1)



1 2 3 4 5 6 7 8

First mid

Last

d) iteration 3 : first<= last and not found

mid=(5+5)div2=5

list (5)=25 so ,so found is true

1 2 3 4 5 6 7 8

Mid

Found is true ,so location ← mid

ان البحث الثنائي يكون فعالا وسريعا في القوائم التي يكون عدد عناصرها n كبيرا اما عندما يكون عدد العناصر قليلا فان البحث التسلسلي يصبح هو الاسرع والشكل التالي يوضح مقارنة بين عدد المقارنات وعدد العناصر لكل من البحث التسلسلي والبحث الثنائي والعلاقة بينهما.

Sequential search O(N)

Binary Search O(log2N)

عدد المقارنات

عدد العناصر(N)

الشكل يوضح مقارنة بين البحث التسلسلي والبحث الثنائي

برنامج ١٩- : تمثيل خوارزمية البحث الثنائي (binary search) باستخدام صيغة التكرار (iteration)

```
#include<iostream.h>
#include<conio.h>
const n=20;
int data[n], flag;
char compare(int x,int y)
{
 if(x>y)
 return('>');
 else
```

```
   {
    if(x<y)    return('<');
    else return('=');
   }
 }
 void binsearch(int a[n],int x,int lower,int upper)
 {
  int j,mid;
  j=0;
  flag=0;
  while((lower<=upper)&&(!flag))
    {
     mid=(lower+upper)/2;
     switch (compare(x,a[mid]))
      {
       case'>':
        {
         lower=mid+1;
         break;
        }
       case'<':
        {
         upper=mid-1;
         break;
        }
       case'=':
        {
         j=mid;
         flag=1;
          break;
        }
      }
    }
 }
 void main()
 {
  int i,m,item;
```

```
 clrscr();
 cout<<"create the main list..how many elements:?"<<endl;
 cin>>m;
 for(i=0;i<m;i++)
  {
   cout<<"enter the item"<<endl;
   cin>>data[i];
  }
 printf("\ninput the value you are looking for\n");
 scanf("%d",&item);
 binsearch(data,item,0,m-1);
 if(flag==1)
 printf("\ngood...the key %d is exist\n",item);
 else
 printf("\nthe key %d is not exist\n",item);
 getch();
}
```

برنامج – ٢٠ : تمثيل خوارزمية البحث الثنائي باستخدام صيغة الاستدعاء الذاتي (recursion).

```
#include<stdio.h>
#include<conio.h>
const size=20;
int list[size];
int key,i,n,location;
int binsearch(int list[size],int lower,int upper,int key)
{
 int x= -1;
 if(lower<=upper)
  {
   x=(lower+upper)/2;
   if(key!=list[x])
    if(key<list[x])
    x=binsearch(list,lower,x-1,key);
    else
    x=binsearch(list,x+1,upper,key);
   }
 return(x);
}
```

```
void main()
{
 int i,n,key1;
 clrscr();
 printf("\nhow many elements in the list\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
  {
   printf("\ninput the element\n");
   scanf("%d",&list[i]);
  }
  printf("\ninput the key you are looking for\n");
  scanf("%d",&key1);
  location=binsearch(list,0,n-1,key1);
  if(location==-1)
  printf("\nthe key is not found\n");
  else
  printf("\nthe key %d is exist at location %d\n",key1,location);
  getch();
}
```

أسئلة الفصل

1-What are the main stages (steps) of any sort process.
2- What are the purposes of the sort?
3- What are the main factors which affect the choice of the sorting process?
4- Perform selection sort algorithm in ascending order on a list of(8)data elements.
5- Name one type of the external sort you know and perform its algorithm on a list of (19) integer values to be sorted in descending order.
6- Write an algorithm for quick sort,perform this algorithm on the key elements:
(90,20,50,80,40,95,70,25,15,17) And show the algorithm iteration.
7- State four types of search algorithms.
8- Write an algorithm for sequential search.
9- Write an algorithm for binary search.

10- Consider the following keys

100 120 150 170 200 250 290 300 310 340

And perform the binary search algorithm to find the key =250.

تقبلوا حبي واحترامي بلال العلواني

وانا مستعد لأي استفسار

من كلية المعارف الجامعة

في قسم علوم الحاسوب

2012-05-26

bilal_z92@yahoo.com